CARLA MERKLE WESTPHALL

UM ESQUEMA DE AUTORIZAÇÃO PARA A SEGURANÇA EM SISTEMAS DISTRIBUÍDOS DE LARGA ESCALA

FLORIANÓPOLIS
2000

UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

UM ESQUEMA DE AUTORIZAÇÃO PARA A SEGURANÇA EM SISTEMAS DISTRIBUÍDOS DE LARGA ESCALA

Tese submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Doutora em Engenharia Elétrica.

CARLA MERKLE WESTPHALL

Florianópolis, Novembro de 2000.

UM ESQUEMA DE AUTORIZAÇÃO PARA A SEGURANÇA EM SISTEMAS DISTRIBUÍDOS DE LARGA ESCALA

Carla Merkle Westphall

'Esta Tese foi julgada adequada para obtenção do Título de Doutora em Engenharia Elétrica, Área de Concentração em *Sistemas de Informação*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.'

Prof. Joni da Silva Fraga, Dr.
Orientador

Prof. Aguinaldo Silveira e Silva, Ph.D.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

Prof. Joni da Silva Fraga, Dr.
Presidente

Prof. Ricardo Dahab, Ph. D.

Prof. Michael Anthony Stanton, Ph. D.

Prof. Carlos Alberto Maziero, Dr.

Prof. Rômulo Silva de Oliveira, Dr.

Prof. Jean-Marie Farines, Dr.

Para *Carlos* e *Johann*...

Agradecimentos

Este trabalho contou com a colaboração de várias pessoas, que não podem deixar de ser lembradas e receber os meus agradecimentos.

Um agradecimento especial ao meu orientador Joni, pela paciência nos estágios iniciais dos trabalhos, pela orientação, pelo apoio e motivação nos momentos de leitura de textos e discussões e por todas as idéias que fizeram o trabalho crescer.

Merecem um obrigado toda a equipe do Projeto *JaCoWeb*. Agradeço a Michelle, pelas conversas, colaboração e implementações realizadas. Agradeço ao Lau, pelas idéias e colaboração nas implementações. Agradeço aos bolsistas Ricardo e Luciana por todo o esforço para tornar concreto e implementado nosso esquema de autorização. Agradeço ao Rafael, pela colaboração nas implementações e pelas críticas que auxiliaram a refinar alguns aspectos deste trabalho. Obrigado também ao Altair pelas contribuições e discussões. Ao Eduardo agradeço pelos favores prestados.

Agradeço também todos os colegas do LCMI pela companhia e apoio.

Obrigada a minha mãe Erica, a quem devo toda a minha formação básica. Agradeço as minhas irmãs pelo incentivo e compreensão.

Ao meu esposo Carlos e ao meu filho Johann, agradeço pela paciência, tolerância e apoio nos momentos difíceis.

Não poderia deixar de agradecer ao povo brasileiro, que permitiu a CAPES financiar a realização deste trabalho.

Obrigado a Deus que iluminou meu caminho e me deu saúde, paz e "segurança" para chegar até aqui.

Resumo da Tese apresentada à UFSC como parte dos requisitos necessários para a obtenção do grau de Doutora em Engenharia Elétrica.

UM ESQUEMA DE AUTORIZAÇÃO PARA A SEGURANÇA EM SISTEMAS DISTRIBUÍDOS DE LARGA ESCALA

Carla Merkle Westphall

Novembro/2000

Orientador: Joni da Silva Fraga, Dr.
Área de Concentração: Sistemas de informação.
Palavras-chave: Segurança, Esquema de Autorização, Políticas de Segurança.
Número de Páginas: 264.

A especificação e o gerenciamento de políticas de autorização em sistemas que usam objetos distribuídos de larga escala representa um desafio para o estabelecimento de ambientes seguros. Esta tese propõe um Esquema de Autorização para sistemas distribuídos usando abstrações do modelo CORBA de segurança integrado aos suportes de segurança das plataformas Java e Web. Este esquema de autorização, construído usando componentes COTS (*Commercial-Off-The-Shelf*), deve servir como um meio de expressão de modelos de segurança como o Matriz de Acesso (controle de acesso discricionário), Bell e Lapadula (controle obrigatório) e modelos Baseados em Papéis (*Role-Based Access Control Models*) em ambientes de larga escala. É feita a proposição de um serviço de política - o *PoliCap* - que supre as carências existentes no CORBAsec no que tange o gerenciamento de políticas de segurança, permitindo expressar as políticas de autorização do ambiente. Um protótipo do esquema de autorização foi implementado em nossos laboratórios e forneceu subsídios para a realização de uma avaliação da segurança do esquema proposto. Uma avaliação da segurança do Esquema de Autorização foi feita com base no padrão ISO 15408 (*Critérios Comuns para Avaliação da Segurança de Tecnologias de Informação*). O esquema proposto foi classificado com o nível EAL 3, de acordo com este padrão.

Abstract of Thesis presented to UFSC as a partial fulfillment of the requirements for the degree of Doctor in Electrical Engineering.

AN AUTHORIZATION SCHEME FOR SECURITY IN LARGE-SCALE DISTRIBUTED SYSTEMS

Carla Merkle Westphall

November/2000

Advisor: Joni da Silva Fraga, Dr.
Area of Concentration: Information systems.
Keywords: Security, Authorization Schemes, Security Policies.
Number of Pages: 264.

The specification and management of authorization policies in systems that use large scale distributed objects represent a challenge for the establishment of secure environments. This thesis proposes an Authorization Scheme for distributed systems using CORBA security model abstractions integrated with Java and Web security platforms. This authorization scheme, built using COTS (*Commercial-Off-The-Shelf*) components, should serve as a way to express security models such as the Access Matrix (discretionary access control), Bell and Lapadula (mandatory access), and *Role-Based Access Control* Models, in large-scale environments. The proposition of a policy service – the *PoliCap* – that meet the need identified in CORBAsec related to policy object management, provides an approach to express the environment's authorization policies. A prototype of the authorization scheme was implemented in our laboratories and it provides subsidies for the accomplishment of a security evaluation of the proposed scheme. A security evaluation of the authorization scheme was performed according to ISO Standard 15408 (*Common Criteria for Security Evaluation of Information Technologies*). The scheme proposed was classified as level EAL 3, based on this standard.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

INTRODUÇÃO

1.1 Motivações

A Internet vêm se caracterizando como uma infraestrutura de comunicação rápida e de baixo custo. Com isto, tem surgido uma crescente demanda por novos serviços e aplicações nestes sistemas distribuídos de larga escala.

Estas novas demandas têm provocado o surgimento de novos paradigmas e ferramentas de programação distribuída. Os componentes COTS (*Commercial-Off-The-Shelf*), definidos como programas e bibliotecas prontos e disponíveis, fazem parte dessas novas tecnologias que estão sendo atualmente utilizadas. Fazer uso de tais componentes pode reduzir substancialmente a complexidade, tempo e custo no desenvolvimento destes sistemas: fica-se livre para escolher componentes COTS mais adequados e atualizados

A segurança sempre foi uma questão importante em sistemas distribuídos, embora tenha sido um pouco desconsiderada inicialmente na evolução das redes de computadores e em sistemas distribuídos. A segurança passou a se firmar como um requisito básico nestes ambientes de larga escala devido ao crescente uso de aplicações críticas como o comércio eletrônico, sistemas de suporte a decisão em grupo e sistemas bancários.

As plataformas Java, CORBA e Web são exemplos de componentes COTS que concretizam o conceito de objetos distribuídos[1] nestes sistemas de larga escala (Carzaniga, Picco, et al. 1997, Ghezzi e Vigna, 1998, Orfali e Harkey, 1997, Resnick, 1996).

As especificações **CORBA** (*Common Object Request Broker Architecture*) foram introduzidas pela OMG[2] como padrões para uma programação distribuída aberta em ambientes heterogêneos (OMG, 1999b). A aceitação destes padrões tem sido ampla nos últimos anos. A linguagem **Java** popularizou o conceito de código móvel através dos seus *applets* executados a partir de *browsers* Web (THORN, 1997). O ambiente Web representa a estrutura mais simples para códigos móveis, disponibilizando toda uma rede mundial e possibilitando carga de códigos em

[1] *Objetos distribuídos* nada mais são do que objetos implementados segundo seus conceitos usuais de programação orientada a objetos, executados em ambientes distribuídos suportando transparência de localização de objetos, invocação de métodos em objetos locais ou remotos e a migração de objetos.

[2] É um consórcio de empresas com o objetivo de especificar um conjunto de padrões e conceitos para a programação orientada a objetos em ambientes distribuídos abertos.

qualquer ponto da rede (GOLLMANN, 1999). O CORBA pode estender o alcance das aplicações Java através de redes e sistemas operacionais. Também dá suporte a aplicações Java com um conjunto rico de serviços (segurança, nomes, transações, etc).

Esses dois padrões de objetos, Java e CORBA, se complementam muito bem (EVANS e ROGERS, 1997, ORFALI e HARKEY, 1997, ORFALI, HARKEY, *et al.* 1997): o CORBA lida com a transparência de rede enquanto Java trata com a independência da plataforma de implementação. O casamento de Java, CORBA e a Web formam juntos uma poderosa ferramenta de desenvolvimento e suporte para aplicações distribuídas, chamado **Web de Objetos** (*Object Web* (ORFALI e HARKEY, 1997)), acessível em qualquer ponto da rede mundial. O modelo de programação distribuída que usa Java, CORBA e Web está se tornando padrão *de facto* de programação na Internet (EVANS e ROGERS, 1997). A combinação da carga automática do código do cliente e, também, a completa independência de plataforma operacional é uma grande vantagem das aplicações que usam Java, CORBA e Web.

Essas novas abordagens introduzem também novos problemas de segurança e também implicam na necessidade de novos conceitos e modelos de segurança diante desse novo quadro (ABRAMS, 1998, GOLLMANN, 1999, LANG, 1997, LANG e SCHREINER, 2000, RUBIN, GEER, *et al.* 1997, SUN, 1998, SUN, 1999, SUN, 2000). Especificamente relacionado com a plataforma Java, um *applet* não confiável apresenta muitas ameaças à segurança (MCGRAW e FELTEN, 1997, SUN, 1997b). Para tratar a segurança da plataforma Java, a empresa Sun Microsystems Inc. desenvolveu **o modelo de segurança Java** (SRINIVAS, 2000, SUN, 1998, SUN, 1999, SUN, 2000).

Além das ameaças impostas pelos *applets*, a complexidade inerente aos sistemas de objetos distribuídos provoca desafios especiais que devem ser tratados pela arquitetura de segurança (LANG, 1997): o controle de acesso em sistemas de larga escala é problemático, já que objetos são constantemente adicionados, removidos e modificados nestes ambientes. Em sistemas geograficamente muito grandes normalmente existe diferentes domínios de políticas de segurança, com diferentes administrações, tornando difícil a interoperabilidade entre estes domínios.

No sentido de minimizar estes problemas a OMG (*Object Management Group*) introduziu o **modelo de segurança do CORBA** (***CORBAsec***) que, quando implementado e administrado corretamente, pode prover um alto nível de segurança para informações e aplicações de objetos distribuídos em ambientes de larga escala (OMG, 2000a).

Esquemas de autorização[3] para esses tipos de sistemas que envolvem mobilidade de código e uso de objetos distribuídos implementados a partir de Java, CORBA e Web, portanto, devem sofrer um processo de renovação e amadurecimento, o que fornece um amplo tema de pesquisa na área de segurança.

Uma arquitetura de segurança para sistemas distribuídos deve tratar das seguintes questões (GOLLMANN, 1999):

- freqüentemente, os requisitos de segurança em sistemas distribuídos excedem a simples autenticação;
- componentes diferentes em um sistema distribuído não necessariamente usam os mesmos mecanismos de segurança. A segurança deve ser garantida acima dessa variedade de mecanismos;
- desenvolvedores e usuários de aplicações não são especialistas em segurança. Deve-se encontrar formas de fornecer serviços de segurança para usuários não conscientes destes serviços.

Além destas questões, a política de segurança também merece destaque. Usualmente nos trabalhos realizados em sistemas distribuídos e redes de larga escala, a política é um pouco desconsiderada ou definida em termos gerais, normalmente colocando as mesmas na forma de políticas discricionárias (definidas pelos proprietários dos objetos a serem protegidos). Entendemos que a definição das políticas ou do tipo de política é importante, e que a modelagem, a especificação e o gerenciamento de políticas de segurança em sistemas distribuídos e ambientes que usam componentes COTS, não é uma tarefa trivial e representa um desafio na pesquisa atual (BROSE, 1998, MARKHAM, COLBY, *et al.* 1999).

1.2 Objetivos da tese

Este trabalho apresenta um esquema de autorização para a programação distribuída em redes de larga escala. Este esquema de autorização integra os suportes de segurança das especificações CORBA e das plataformas Java e Web, visando a implantação de políticas de segurança nestes ambientes.

Um dos objetivos desse trabalho é mostrar que este esquema de autorização, montado a partir de COTS, pode ser um meio de expressão de modelos como o Matriz de Acesso (controle de acesso discricionário), Bell e Lapadula (controles obrigatórios) e modelos Baseados em Papéis (*Role-Based Access Control Models*) em ambientes heterogêneos e de larga escala.

[3] O *esquema de autorização* é a concretização da política de autorização através de um conjunto de mecanismos de segurança.

Quando surgiu a idéia da integração destas ferramentas e o trabalho começou a ser delineado, começamos também a tomar pé das dimensões do mesmo. Um conjunto de questões então surgiu, definindo os desafios que teríamos pela frente. Estas questões, envolvendo o desenvolvimento de esquemas de autorização em redes de larga escala, podem ser resumidas em:

- Como definir e gerenciar políticas de segurança em redes de larga escala como a Internet, garantindo a programação distribuída segura de forma transparente aos usuários ? Será possível a partir de um esquema de autorização, concebido a partir de componentes COTS, implementar políticas ou modelos de segurança em redes de larga escala ?

- Como fornecer suporte adequado para desenvolvedores de aplicação e administradores de segurança para o projeto, especificação e implementação de políticas de segurança em sistemas de objetos distribuídos ?

- Como avaliar um esquema de autorização ? Qual a forma de conduzir uma avaliação deste tipo ?

Esperamos que o trabalho desenvolvido e apresentado nesta tese ajude a responder a estas perguntas. O esquema de autorização proposto nesta tese define um serviço de política que lida com a implantação de políticas de segurança discricionárias e obrigatórias, em ambientes que conjugam os suportes de segurança do CORBA, Java e Web.

1.3 Organização do texto

O texto dessa tese reflete diversas etapas que foram cumpridas durante o trabalho de doutorado. Uma primeira parte espelha uma etapa de amadurecimento na qual foi necessário o levantamento do estado da arte nas disciplinas de segurança e suportes de segurança existentes. Nas outras partes é feita a apresentação e avaliação do esquema de autorização proposto e implementado.

Este trabalho é dividido em sete capítulos. Este primeiro capítulo descreveu o contexto no qual o trabalho de doutorado está inserido e apresentou os objetivos da tese.

O capítulo 2 apresenta os conceitos principais relacionados com a segurança em sistemas informáticos. São discutidas políticas de autorização e a implementação dos esquemas correspondentes em sistemas distribuídos. Os modelos de segurança apresentados são importantes no entendimento da dinâmica associada com as políticas correspondentes. O estudo descrito nesse capítulo coloca as abordagens que podem ser adotadas na condução do tratamento da segurança em sistemas distribuídos.

O terceiro capítulo apresenta a segurança em sistemas distribuídos abertos. Um enfoque especial é dado ao suporte de segurança das especificações CORBA, por constituir uma infraestrutura muito poderosa para a definição de esquemas de autorização. A segurança das plataformas Java e Web também é discutida. O ambiente Web fornece um suporte também poderoso e útil para programação distribuída das mais diversas aplicações em um contexto de redes de larga escala. A visão geral da proposta do Esquema de Autorização *JaCoWeb* é apresentada neste capítulo, definindo as funcionalidades de segurança que compõem esta arquitetura de segurança.

O capítulo 4 apresenta a discussão da implementação de políticas discricionárias no Esquema de Autorização *JaCoWeb*, apresentando os dois níveis de controle de acesso do esquema. O nível global, representado pelo serviço de política *PoliCap* proposto, concretiza o primeiro nível de controle de acesso do esquema de autorização. O segundo nível de controle de acesso é desenvolvido com o uso de *capabilities* propostas para o modelo CORBA de segurança. O capítulo ainda apresenta e discute a implementação do Esquema de Autorização Discricionário.

O quinto capítulo explora a discussão sobre a implementação de políticas obrigatórias no Esquema de Autorização *JaCoWeb*. Esta proposta consiste em adicionar às estruturas discricionárias do modelo CORBA de segurança controles obrigatórios. O esquema de autorização obrigatório adicionado tem como base o modelo Bell e Lapadula. Essas mesmas estruturas do *JaCoWeb* são também usadas na implementação de políticas baseadas em papéis (*Role-Based Access Control Models*).

O capítulo 6 apresenta a avaliação da segurança do esquema de autorização proposto. O padrão ISO 15408, também chamado Critérios Comuns para a Avaliação da Segurança de Tecnologias de Informação, é usado como base desta avaliação. A metodologia de avaliação e os resultados finais obtidos também são apresentados.

Finalmente, no capítulo 7 são apresentadas as principais conclusões e perspectivas futuras para esse trabalho.

Capítulo 2

SEGURANÇA EM SISTEMAS DISTRIBUÍDOS

2.1 Introdução

Em sistemas distribuídos, a proteção da informação é uma tarefa difícil de ser realizada já que a proteção física dos computadores que contêm informações estratégicas dificilmente é garantida. Além disso, tecnicamente, qualquer pessoa pode ter acesso ao sistema através do suporte de comunicação.

Em sistemas distribuídos de larga escala, construídos sobre redes como a Internet, estes problemas se multiplicam. A heterogeneidade, o grande número de usuários, objetos e operações dificultam a definição de políticas de segurança nestes sistemas. As técnicas de gerência de segurança apresentam deficiências de escalabilidade quando tratam com a complexidade destas redes (BLAKLEY, 1996).

Este capítulo introduz a discussão sobre a definição de políticas de segurança em sistemas distribuídos. Inicialmente, apresenta alguns conceitos essenciais sobre segurança e descreve as principais violações presentes em sistemas distribuídos. Mostra também a importância da definição de políticas de segurança e, para isto, apresenta os modelos de segurança presentes na literatura. Na seqüência são discutidas técnicas de implementação de mecanismos de segurança em sistemas distribuídos.

2.2 Conceito de Segurança

O conceito de Segurança (*"SECURITY"*) em um sistema informático é identificado com a sua capacidade de assegurar a prevenção ao acesso e à manipulação ilegítimos da informação ou, ainda, de evitar a interferência indevida na sua operação normal (ISO/IEC 15408-1, 1999).

Esta capacidade está fundamentada sobre quatro propriedades que devem ser mantidas (KENT, 1977, STALLINGS, 1998):

- A **confidencialidade** garante que as informações serão disponíveis somente aos usuários autorizados às mesmas;

- A **disponibilidade** deve garantir sempre o acesso autorizado às informações. O serviço de um sistema não pode ser interrompido por ações ou conhecimento de indivíduos não autorizados;
- A **integridade** está ligada à capacidade do sistema de impedir a corrupção das informações por faltas intencionais ou acidentais;
- A **autenticidade** está ligada a meios que garantem a validade da informação, tanto dados quanto informações de usuários, num dado instante.

Normalmente, as três primeiras propriedades são encontradas na literatura. A autenticidade é incluída na lista, quando existe preocupação quanto aos aspectos da comunicação no ambiente. Outra propriedade que também pode fazer parte dessa lista é a contabilização (*accounting*), principalmente quando o interesse recai em aplicações como o comércio eletrônico (GOLLMANN, 1999).

Garantir a segurança de um sistema é então uma tarefa difícil de ser executada diante das dimensões dos sistemas atuais. É necessário, para tal, conhecer todos os caminhos que pode percorrer ou estar disponível uma informação no sistema. É um trabalho que pode ser extremamente complexo e com custos bem além das necessidades reais dos usuários do sistema.

2.3 Violações de Segurança

As informações em um sistema informático são susceptíveis a três categorias de violações de segurança:

- Revelação não autorizada de informação;
- Modificação não autorizada de informação;
- Negação de serviço.

A primeira categoria de violação envolve a não verificação da propriedade de confidencialidade. As violações do tipo **modificação não autorizada de informação** atentam contra a integridade das informações estocadas ou veiculadas no sistema.

As violações do tipo **negação de serviço** têm como alvo o impedimento do acesso legítimo a recursos de um usuário autorizado. Isso pode, por exemplo, tornar um recurso não disponível para usuários legítimos através de uma carga computacional alta gerada no sentido de tornar o serviço indisponível.

Na prática, a concepção de sistemas onde essas violações sejam evitadas no todo é muito difícil. Os problemas de confinamento definidos por Lampson (LAMPSON, 1973, MILLEN, 1999) são prova da dificuldade de evitar a revelação não autorizada de informação. Todo sistema onde a concepção está baseada no compartilhamento de recursos, apresenta possibilidades sutis e não

esperadas para a transferência de informação entre duas entidades não autorizadas a se comunicar (canais de memória e canais cobertos).

Na literatura é também utilizado o termo **ameaça** para definir ou identificar circunstâncias, condições ou eventos que forneçam algum potencial de violação de segurança. Uma **vulnerabilidade** é uma falha ou característica indevida que pode ser explorada para concretizar uma ameaça. Atualmente, há um esforço chamado de *CVE* (*Common Vulnerabilities and Exposures*), no centro de segurança da empresa americana MITRE, no sentido de estabelecer uma nomenclatura padronizada para cada tipo de vulnerabilidade existente. Várias empresas já estão adaptando seus produtos e bancos de dados de acordo com os termos estabelecidos pelo CVE (WILLIAMS, 2000).

Um **ataque** é identificado como um conjunto de ações conduzidas por uma entidade não autorizada visando violações de segurança. Os ataques são tradicionalmente classificados em **ataques passivos** e **ativos**. Alguns exemplos de ataques típicos em sistemas distribuídos e redes de computadores são descritos na Tabela 2.1. Ataques passivos podem envolver a monitoração sem a alteração da informação em trânsito, por exemplo. Ataques passivos, portanto, ameaçam a confidencialidade dos dados. Os ataques ativos envolvem a modificação não autorizada, a negação de serviço e a fabricação de objetos espúrios a serem inseridos no sistema, afetando como conseqüência a integridade, a disponibilidade e a autenticidade das informações, respectivamente (STALLINGS, 1998).

Ameaça	Descrição
Análise de tráfego (ataque passivo)	É uma ameaça envolvendo a escuta de linha, onde as informações disponíveis são os campos de controle do protocolo de comunicação. Pode envolver o uso de softwares chamados *sniffers*.
Trapdoor (ataque ativo ou passivo)	Estas ameaças são constituídas de mecanismos aparentemente legítimos, construídos de maneira que, na apresentação de dados específicos, permitem o acesso à informação sem observar os controles habituais.
Cavalo de Tróia (ataque ativo ou passivo)	Essa ameaça é constituída por *softwares* que executam, além de operações descritas em suas especificações e disponíveis a seus usuários, operações invisíveis em favor do proprietário. Cavalo de tróia faz parte dos problemas de confinamento e é muito difícil de tratar.
Resource stealing (ataque ativo)	Um recurso (por exemplo, uma porta de acesso) é usada de forma tão intensa que o serviço para outros usuários fica indisponível. A violação correspondente é a negação de serviço. Um exemplo atual desse tipo de ataque são os chamados ataques DDoS (*Distributed Denial of Service*), que deixou os *sites* Yahoo.com, Amazon.com e CNN.com fora do ar em Fevereiro de 2000 (GARBER, 2000).
Interceptação/ Alteração (ataque ativo)	Uma informação em transferência pode ser interceptada, removida e modificada. Um exemplo de ataque associado a essa ameaça é o *IP Spoofing* que consiste na troca de endereço IP original por outro, podendo dessa maneira um *host* se fazer passar por outro.
Replay ou ataque por mensagem antiga (ataque ativo)	Uma cópia capturada da mensagem (ou informação) transmitida anteriormente, é retransmitida com objetivos ilegítimos.
Serviço malicioso (ataque ativo)	Um sistema ou componente do sistema falso é construído no sentido de enganar usuários legítimos ou sistemas para obter informações sensíveis ou não autorizadas.

Tabela 2.1 – Ataques típicos em redes de computadores (FORD, 1994).

A maioria dos ataques bem sucedidos aos sistemas de computação via Internet, atualmente, aproveitam a existência de um pequeno número de ameaças bem conhecidas entre os invasores e até mesmo entre administradores de redes. Em junho de 2000, através de uma cooperação entre a indústria, o governo e a comunidade acadêmica, foram estabelecidas as dez ameaças mais críticas à segurança da Internet (SANS, 2000).

2.4 Políticas de Segurança

Com o objetivo de impedir as violações de segurança e assegurar que o conjunto das propriedades fundamentais seja mantido é definida a **política de segurança** de um sistema.

A **política de segurança** de um sistema é o conjunto de regras e práticas que determinam a maneira pela qual as informações e recursos são gestionados, protegidos e distribuídos no interior de um sistema específico (ISO/IEC 15408-1, 1999).

A noção de política de segurança pode ser refinada em dois grupos distintos (FRAGA, 1985, NICOMETTE, 1996): as **políticas de segurança física** e **políticas de segurança lógica**.

As políticas de segurança física se ocupam com tudo que se refere à situação física do sistema a proteger. Deve-se notar que no caso de sistemas distribuídos, a segurança física torna-se cada vez menos eficaz devido ao número significativo de pontos de acesso ao sistema (estações e meios de comunicação). A expansão da Internet, cada vez com mais pessoas conectadas a partir de seus domicílios, graças a um computador pessoal, é a prova disto.

Políticas de segurança lógica tratam das propriedades de segurança em tempo de execução no sistema computacional, sendo concretizadas por controles (internos) lógicos, especificando "quem" tem acesso a "que" e em "quais" circunstâncias.

A política de segurança lógica pode ser decomposta em várias fases. Cada indivíduo que utiliza um sistema seguro deve se identificar e deve poder provar que ele realmente é a pessoa que pretende acessar um recurso ou informação do sistema. Estas duas idéias são definidas como **política de identificação** e **política de autenticação**. Quando um usuário é identificado e autenticado, a **política de autorização** ou **política de controle de acesso** (SHANDU, 1994) deve especificar quais são as operações que este usuário particular pode realizar no sistema.

O **esquema de autorização** é a concretização da política de autorização através de um conjunto de mecanismos de segurança. Um esquema de autorização pode ser representado em termos de estados de segurança do sistema e de regras de transição que permitem alterar o estado de proteção do sistema. Por exemplo, uma regra do esquema de autorização pode ser "o proprietário de uma informação pode conceder um direito de acesso para esta informação a qualquer usuário".

Se a política de autorização é dita **coerente**, então, não é possível partir de um estado inicial seguro (estado onde as propriedades de segurança são mantidas) chegar a um estado inseguro (onde as propriedades não são satisfeitas), ao se aplicar as regras do esquema de autorização correspondente.

As entidades que são responsáveis e autorizadas pela política de segurança a efetuar acessos sobre informações mantidas pelo sistema são denominados de **principais**. A granularidade da aplicação da política pode determinar o principal como um usuário, um processo ou ainda uma máquina hospedeira (*host*) numa rede. As entidades não autorizadas ou **intrusos** podem ser identificadas como usuários, processos, máquinas hospedeiras, etc. Em um ataque o intruso usa os seus conhecimentos sobre ameaças presentes no sistema.

As políticas de autorização ou os esquemas de autorização se classificam em duas categorias: políticas discricionárias e políticas obrigatórias ou não-discricionárias.

No caso das **políticas discricionárias**, os direitos de acesso a cada recurso, a cada informação, são manipulados livremente pelo responsável do recurso ou da informação (geralmente o proprietário do mesmo), segundo a sua vontade (à sua **discrição**). A gestão do acesso aos arquivos Unix constituem um exemplo de controles baseados em políticas discricionárias. O proprietário de um arquivo, nesse caso, pode atribuir livremente ou não os direitos (ler, escrever, executar, etc) a ele mesmo, a um grupo de usuários e a outros usuários. As políticas discricionárias não são coerentes.

As políticas ditas **obrigatórias** ou **não-discricionárias** resumem em seus esquemas de autorização um conjunto de regras incontornáveis que expressam um tipo de organização envolvendo a segurança das informações no sistema como um todo. A política obrigatória supõe os usuários e os objetos ou recursos do sistema todos etiquetados; as etiquetas dos objetos seguem uma classificação específica enquanto os usuários ou os sujeitos do acesso possuem níveis de habilitação. Controles que determinam as autorizações de acesso são baseados numa comparação da habilitação do usuário com a classificação do objeto. As regras definidas nesses controles e que são ditas como incontornáveis asseguram que o sistema verifique as propriedades de confidencialidade e de integridade.

A política obrigatória do DoD (*Department of Defense* – Departamento de Defesa dos Estados Unidos) formalizada por Bell e Lapadula (BELL e LAPADULA, 1976, LANDWEHR, 1981) é um dos exemplos dessas políticas obrigatórias, válidas para todo o sistema considerado. Os esquemas de políticas obrigatórias são utilizados conjuntamente com os discricionários. Nesse caso, um usuário é autorizado a manipular uma informação se possui o direito ao acesso

correspondente (controle discricionário) e se é habilitado ao nível de classificação da informação (controle obrigatório).

2.5 Modelos de Segurança

Os **modelos de segurança** são formas de descrever as políticas de autorização, determinando tanto o comportamento de entidades governadas pela política quanto as regras que definem a evolução desta política (GOLLMANN, 1999).

Modelos matemáticos formais de segurança, quando usados para descrever políticas de autorização (LANDWEHR, 1981), permitem, de certa maneira, verificações de que a política é coerente, e servem de guia para implementações de esquemas de autorização correspondentes às especificações contidas no modelo.

Os modelos de segurança descrevem políticas que têm como ponto principal manter a propriedade da confidencialidade, como o modelo Bell e Lapadula, ou a propriedade da integridade, como os modelos de Biba e Clark-Wilson. A formalização do modelo Bell e Lapadula, por exemplo, tem destaque em avaliações de segurança de sistemas que o adotam como base para políticas obrigatórias (GOLLMANN, 1999). Modelos informais, como o Clark-Wilson, constituem-se em *frameworks* descritivos para expressar políticas de autorização.

2.5.1 Matriz de Acesso

A noção de matriz de acesso foi introduzida por Lampson (LAMPSON, 1971) como uma generalização dos mecanismos de proteção usualmente fornecidos por sistemas operacionais. A estrutura descrita nesse modelo serve para descrever uma grande variedade de mecanismos de segurança. A formulação desse modelo simples e geral está fundamentada sobre as noções de **sujeitos**, **objetos** e de **direito**. Um sujeito é uma entidade ativa se executando em favor de um usuário (por exemplo, um processo Unix). Um objeto é considerado como uma entidade passiva que é definida por um estado e operações de acesso. Em um dado instante, um sujeito tem o direito de acesso sobre um objeto podendo portanto, executar a operação correspondente sobre o objeto, se esse direito estiver expresso na matriz de acesso. Essa matriz representa a configuração corrente da proteção do sistema.

O modelo matriz de acesso representando a configuração corrente da segurança do sistema, possui como linhas na matriz os sujeitos e colunas os objetos. A intersecção de uma linha com uma coluna determina o conjunto de direitos de acesso que o sujeito possui sobre o objeto correspondente. Esses direitos constituem o conjunto de operações que o sujeito pode executar sobre o objeto. A proteção implementada por esse modelo consiste em definir o número de direitos

de cada sujeito sobre cada objeto e garantir que somente os acessos correspondentes a esses direitos serão executados no sistema. A Figura 2.1 sintetiza o modelo matriz de acesso.

Objetos

Sujeitos		**Arquivo joao.doc**	**Arquivo edit.exe**	**Procedimento P**	**Segmento S**
	Domínio D_1	ler, escrever	ler	entrar	ler, escrever
	Domínio D_2		ler, escrever	entrar	
	Alice		executar		
	Joao	ler, escrever	executar		

Figura 2.1 – Modelo matriz de acesso.

Uma matriz de acesso não é fixa, mas evolui no tempo em função da criação de novos objetos e sujeitos. A evolução da matriz de acesso no tempo é definida em termos de estados de segurança e de regras de transição. A matriz de acesso corresponde à representação do seu estado atual de segurança do sistema. As mudanças de estado de um sistema são realizadas usando regras de transição do modelo. Essas regras são tipicamente: remover objeto/sujeito, criar objeto/sujeito, transferir direitos e suprimir direitos.

As regras do modelo definem operações (acessos) que manipulam o objeto matriz de acesso (a própria matriz é vista como objeto nesse modelo); a matriz deve, portanto, também conter os direitos correspondentes a essas regras. Alguns sistemas concretizam as condições de execução dessas regras em um título de proprietário. O modelo, nesse caso, é discricionário; o sujeito que tem o título de propriedade sobre um objeto, determina a política de utilização desse objeto, podendo distribuir e revogar os direitos de acesso ao objeto que ele criou.

Diante desse aspecto dinâmico, referente à evolução da própria matriz de acesso, um sujeito pode ter acesso a um objeto em duas situações:

- sujeito tem, no seu domínio, o direito referente ao acesso do objeto, no estado atual de segurança do sistema;
- sujeito pode obter esse direito por meio de mudanças do estado de segurança do sistema.

Um estado não autorizado (ou estado de fuga) é um estado onde o direito de acesso pode ser obtido por um sujeito não autorizado. Um sujeito, não autorizado pelo proprietário do objeto, pode ter acesso ao mesmo desde que ele tenha obtido o direito correspondente nas evoluções da matriz de acesso.

Várias representações formais e gráficas da matriz de acesso foram propostas no sentido de verificar o comportamento dinâmico do modelo matriz de acesso. Os modelos HRU (HARRISON, 1976), Take-Grant (SNYDER, 1981) são exemplos desses esforços que visam prever os possíveis estados não autorizados de um sistema.

Na verdade, o modelo matriz de acesso é uma representação abstrata de políticas discricionárias, e portanto não coerentes. As regras especificadas nesse modelo são gerais, o que determina a grande aplicabilidade e a flexibilidade do mesmo. Essa generalidade complica a prova de asserções referentes a coerência das políticas e esquemas de autorização construídos sobre esse modelo (HARRISON, 1976).

2.5.2 Modelos Baseados em Lattices (Reticulados)

Em organizações públicas ou privadas, existem certas informações sigilosas que se forem descobertas por pessoas não autorizadas podem comprometer a própria existência destas organizações. Como há custos associados à proteção da informação e nem todas as informações são igualmente importantes (ou *sensíveis*), são definidos **níveis de segurança**, ordenados segundo uma hierarquia. Os níveis mais usuais, em ordem crescente de sensibilidade, são definidos como: NÃO-CLASSIFICADO, CONFIDENCIAL, SECRETO e ULTRA-SECRETO (DOD, 1985).

Além dos níveis de segurança, são definidas **categorias de segurança**, ou compartimentos de segurança, que correspondem a diferentes projetos, setores ou departamentos da organização. Os indivíduos têm acesso a diferentes categorias na medida em que as suas incumbências demandem este acesso. Assim, por exemplo, clientes de uma agência bancária provavelmente não devem ter acesso a informações classificadas como ULTRA-SECRETO pertencentes ao departamento de gerência deste banco.

No contexto computacional, um **rótulo de segurança** é um atributo que denota a sensibilidade das entidades em um sistema e é composto por um **nível de segurança** e por um conjunto (possivelmente vazio) de **categorias de segurança**. Em sistemas que fazem uso deste mecanismo, todas as entidades recebem um rótulo de segurança; o rótulo de um objeto é chamado de **classificação** do objeto, e o rótulo de um sujeito é chamado de **habilitação** (*clearance*) do sujeito.

Os modelos baseados em reticulados formalizam esquemas de autorização não discricionários. A definição de reticulados é apresentada no quadro abaixo.

Definição 2.1 Um reticulado (*lattice*) consiste de um conjunto L e de uma ordem parcial $\leq$ tal que para cada dois elementos $a, b \in L$, existe um menor limite superior ou supremo u L e, um maior limite inferior ou ínfimo $l \in L$, isto é,

$$a \leq u,\ b \leq u, \text{ e para todo } v \in L : (a \leq v \wedge b \leq v) \Rightarrow (u \leq v)$$

$$l \leq a,\ l \leq b, \text{ e para todo } k \in L : (k \leq a \wedge k \leq b) \Rightarrow (k \leq l)$$

A relação de ordem parcial definida para o conjunto L de rótulos de segurança que é usada aqui é a relação de **dominância**. Esta relação define que *rótulo1* (*nível1, categoria1*) $\leq$ *rótulo2*

(*nível2, categoria2*) (lê-se *rótulo1* **é dominado** por *rótulo2* ou, *rótulo2* **domina** *rótulo1*) se, e somente se, *nível1* ≤ *nível2* e *categoria1* ⊂ *categoria2*.

Exemplo 2.1 Sejam os seguintes conjuntos

Níveis de Segurança = {CONFIDENCIAL, SECRETO}
Categorias = { ∅, {Exército}, {Marinha}, {Exército, Marinha}}
L = {(CONFIDENCIAL, ∅), (CONFIDENCIAL, {Exército}),
(CONFIDENCIAL, {Marinha}), (CONFIDENCIAL, {Exército, Marinha}),
(SECRETO, ∅), (SECRETO, {Exército}), (SECRETO, {Marinha}),
(SECRETO, {Exército, Marinha})}

Observando o exemplo 2.1, podemos afirmar que (CONFIDENCIAL, ∅) ≤ (SECRETO, {Exército}), isto é, que o segundo rótulo domina o primeiro pois CONFIDENCIAL ≤ SECRETO e ∅ ⊂ {Exército}.

Em um conjunto *L* de rótulos de segurança, o *ínfimo* é representado pelo elemento composto pelo nível mais baixo de segurança e por um conjunto *Categorias* vazio de categorias de segurança. O *supremo* em um conjunto *L* é composto pelo nível mais alto de segurança e por todas as categorias de segurança (o próprio conjunto *Categorias*). Por exemplo, considerando o conjunto *L* do exemplo 2.1, tem-se que ínfimo(*L*) = (CONFIDENCIAL, ∅) e supremo(*L*) = (SECRETO, {Exército, Marinha}).

O conjunto de rótulos de segurança *L* sempre possui um ínfimo e um supremo, portanto, pode-se dizer que o conjunto de rótulos de segurança ordenado parcialmente pela relação de dominância forma um reticulado de rótulos de segurança (LANDWEHR, 1981). Esta conclusão forma o princípio fundamental de diversos modelos de segurança como o modelo de fluxo de informação de Denning (DENNING, 1976) e os modelos Bell e Lapadula e Biba (LANDWEHR, 1981).

2.5.2.1 Bell e Lapadula

No início da década de 70, o governo dos Estados Unidos financiou diversas pesquisas sobre modelos de segurança. Dois cientistas da MITRE Corporation, David Bell e Leonard LaPadula, desenvolveram um modelo baseado nos procedimentos usuais de manipulação de informação em áreas ligadas à segurança nacional americana. Esse modelo ficou conhecido como **modelo Bell e Lapadula**, ou modelo BLP (BELL e LAPADULA, 1976, LAPADULA e BELL, 1996), e é provavelmente o modelo de segurança mais citado.

Existem várias descrições do modelo disponíveis na literatura, como (LANDWEHR, 1981, NICOMETTE, 1996, SANDHU, 1993), algumas delas apresentando pequenas variações em relação ao modelo original.

A idéia chave desse modelo é acrescentar controles de acesso **obrigatórios** aos controles de acesso discricionários, de forma a aplicar (*enforce*) políticas de segurança que impeçam o fluxo de informações de níveis de segurança mais altos para os níveis de segurança mais baixos (SANDHU, 1993). As permissões de acesso são definidas através de uma **matriz de acesso** e **de rótulos de segurança**. A matriz de acesso armazena os direitos de cada sujeito sobre os objetos do sistema e pode ser modificada pelos sujeitos através de regras específicas.

Os esquemas de políticas obrigatórias são utilizados conjuntamente com os discricionários. Nesse caso, um usuário é autorizado a manipular uma informação se possui o direito ao acesso correspondente (controle discricionário) e se é habilitado ao nível de classificação da informação (controle obrigatório).

A cada sujeito do sistema são associados duas etiquetas: a primeira (estática) é o nível de segurança máxima do sujeito, composto pelo nível de habilitação do sujeito (não classificado, confidencial, secreto ou ultra secreto) e sua categoria; a segunda etiqueta é o nível de segurança corrente. O nível de segurança máximo representa o nível máximo de segurança das informações que o sujeito pode consultar. O nível corrente representa o mais alto nível das informações que o sujeito consultou no sistema: este último nível flutua, portanto, com a dinâmica do sistema.

Descrição do BLP

Considerando os seguintes conjuntos (BELL e LAPADULA, 1976):

- Conjunto de **sujeitos** S;
- Conjunto de **objetos** O;
- Conjunto de **operações de acesso** A = { *read, write, append, execute* }

  Onde: *read* : representa o acesso com observação sem modificação
  write : representa o acesso com observação e com modificação
  append[4] : representa o acesso sem observação com modificação (também chamado *blind write*)
  execute : representa o acesso sem observação e sem alteração

- Conjunto L de **rótulos de segurança** ordenados segundo a ordem parcial $\leq$..

O modelo BLP é representado sob a forma de uma máquina de estados com regras de transição captando a dinâmica do modelo, representando os controles obrigatórios sobre os quais os usuários não têm controle. O conjunto de estados deve armazenar todas as permissões e todas as

[4] Em (LAPADULA e BELL, 1996), que é uma revisão do modelo original descrito em (BELL e LAPADULA, 1976), afirma-se que os tipos de acesso *append* e *execute* são opcionais em um sistema de computação que pode ser concebido apenas com acessos do tipo *read* e *write*.

instâncias correntes de sujeitos acessando objetos. Os estados do BLP são representados pelo conjunto $B \times M \times F$ onde:

- $B = P(S \times O \times A)$ é o conjunto dos acessos correntes. Um elemento $b \in B$ é uma coleção de tuplas (s, o, a), indicando que o sujeito s está executando a operação a sobre o objeto o.
- M é o conjunto de matrizes de acesso $M = (M_{so})_{s \in S, o \in O}$.
- $F \subset L^S \times L^S \times L^O$ é o conjunto de rótulos de segurança. Um elemento $f \in F$ é uma tripla (f_S, f_C, f_O), onde
  - $f_S : S \rightarrow L$ é uma função que fornece a ***habilitação*** (nível de segurança máximo) de cada sujeito;
  - $f_C : S \rightarrow L$ é uma função que fornece o ***nível de segurança corrente*** de cada sujeito;
  - $f_O : O \rightarrow L$ é uma função que fornece a ***classificação*** de todos os objetos.

O nível de segurança corrente não pode ser maior que o seu nível de segurança máximo (o seu nível de habilitação), portanto $f_C \leq f_S$, em outras palavras "f_S domina f_C". O nível de segurança corrente serve para possibilitar a mudança de níveis de segurança aos sujeitos.

Propriedades do Modelo

O modelo Bell e LaPadula define duas propriedades básicas: a propriedade simples e a propriedade estrela (BELL e LAPADULA, 1976).

Propriedade simples: também conhecida como *propriedade-ss* ou regra *no read up*[5] (NRU), define que um sujeito só pode ler objetos cujos níveis de segurança sejam **dominados** pelo seu nível de segurança (seu nível de habilitação). Por exemplo, uma informação classificada como SECRETO só pode ser lida por sujeitos com habilitação SECRETO ou ULTRA-SECRETO.

> **Definição 2.2 (Propriedade-ss)** Um estado (b, M, f) satisfaz a propriedade-ss, se para cada elemento $(s, o, a) \in b$ onde a operação de acesso é *read* ou *write*, o nível de segurança do sujeito s domina a classificação do objeto o, isto é, $f_O(o) \leq f_S(s)$.

A propriedade-ss tem por objetivo impedir os usuários de ter acesso a informações às quais eles não são autorizados, mas não é suficiente para garantir a segurança do sistema: esta propriedade não evita que um sujeito malicioso coloque informações privilegiadas em um recipiente com classificação inferior à das informações – o que constitui claramente um fluxo não-autorizado de informação. Assim, torna-se necessário adicionar outra propriedade a ser satisfeita pelo sistema.

[5] *No read up* vem do fato de um sujeito não poder ler objetos localizados acima dele no reticulado de rótulos de segurança.

Propriedade estrela: também conhecida como *propriedade-** ou regra *no write down*[6] (NWD), define que um sujeito pode ler somente objetos **dominados** pelo seu nível corrente de segurança e pode escrever em objetos que **dominem** seu nível corrente de segurança. Por exemplo, se um sujeito está lendo (observando) um objeto SECRETO, ele só pode escrever (modificar) um objeto SECRETO ou ULTRA-SECRETO. A propriedade-* pode ser refinada em termos do nível de segurança corrente de um sujeito, conforme mostra a definição 2.3.

> **Definição 2.3 (Propriedade-*)** Um estado (b, M, f) satisfaz a propriedade-*, se para cada elemento $(s, o, a) \in b$ onde a operação de acesso a é *append* ou *write*, o nível de segurança corrente do sujeito s é dominado pela classificação do objeto o, isto é, $f_C(s) \leq f_O(o)$. Além disso, se existe um elemento $(s, o, a) \in b$ onde a operação de acesso a é *append* ou *write*, então deve existir $f_O(o') \leq f_O(o)$ para todos os objetos o' com $(s, o', a') \in b$ onde a operação de acesso a' é *read* ou *write*.

A propriedade estrela visa impedir o fluxo de informações de um nível para níveis inferiores de segurança no sistema. Algumas observações importantes a respeito da propriedade-* são feitas na seção sobre as limitações do modelo BLP.

Os aspectos discricionários do modelo BLP são expressos por uma matriz de controle de acesso e pela *propriedade-ds*[7] (definição 2.4).

> **Definição 2.4 (Propriedade-ds)** Um estado (b, M, f) satisfaz a propriedade-ds, se para cada elemento $(s, o, a) \in b$ tem-se $a \in M_{SO}$.

O modelo Bell e LaPadula define regras para mudanças de estado. Essas regras são constituídas por operações do tipo transferir acesso, revogar acesso, criar/destruir objetos etc. A execução dessas regras (que modificam a matriz de acesso) devem a *priori* preservar as propriedades definidas no modelo. Um estado é chamado seguro, se **todas as três propriedades** (propriedade-ss, propriedade-* e propriedade-ds) forem satisfeitas.

Limitações do Modelo Bell e LaPadula

O modelo Bell e LaPadula, quando foi proposto, representou um avanço significativo ao definir formalmente conceitos de segurança militar aplicável a sistemas computacionais. Ele serviu de base para diversos esforços de projeto e implementação. Em parte devido a estes esforços, algumas limitações do modelo foram descobertas e são discutidas na literatura. Algumas descrições serão abordadas neste trabalho, mas discussões mais completas podem ser encontradas em (FOLEY, GONG, *et al.*, 1996, LANDWEHR, 1981, MCLEAN, 1990, SANDHU, 1993).

[6] Chamada *No write down* porque impede que um sujeito escreva em objetos localizados abaixo dele no reticulado de rótulos de segurança.

[7] Normalmente esta propriedade é esquecida na literatura, embora conste no modelo original de (BELL e LAPADULA, 1976).

Por exemplo, considerando que o rótulo corrente de um sujeito f_C = SECRETO e que ele deseja ler um objeto *o1*, que possui f_{O1} = CONFIDENCIAL, para realizar uma cópia desse arquivo. A propriedade-* impõe que a classificação f_{O1} da cópia seja superior ou igual a SECRETO, mesmo que as informações ali contidas possuam classificação confidencial. Ao longo do tempo, isso faz com que as informações **subam** no reticulado de rótulos de segurança, recebendo classificações sucessivamente maiores. Este fenômeno é conhecido como **superclassificação da informação** (LANDWEHR, 1981, NICOMETTE, 1996), sendo resultado do aspecto extremamente restritivo da propriedade-*, e representa um dos principais problemas do modelo BLP.

A superclassificação da informação provoca a necessidade de reclassificações periódicas dos objetos, que pode ser feito através dos sujeitos de confiança, apenas para garantir a usabilidade de sistemas baseados no modelo BLP.

Um **sujeito de confiança** (*trusted subject*) é aquele que recebe confiança suficiente de que suas ações não quebrarão a segurança do sistema, mesmo que alguns dos seus acessos violem a propriedade-*. Neste caso, a propriedade-* só se aplica aos demais sujeitos do sistema. Por exemplo, o conceito de sujeitos de confiança pode ser usado para qualificar os processos relacionados com a manutenção do sistema, pois, se o administrador do sistema tiver que obedecer estritamente as regras do modelo BLP, ele dificilmente conseguirá realizar qualquer tarefa significativa de administração. O mais difícil é saber determinar quais os processos que podem ser considerados de confiança (LANDWEHR, 1981) e também saber usar com presteza e cuidado o adjetivo *de confiança* (GOLLMANN, 1999).

O termo ***blind write*** ou *escrita cega* demonstra uma situação gerada pela propriedade-* do modelo BLP, que permite a um sujeito escrever em um objeto com uma classificação superior à sua habilitação mas, proibe a esse mesmo sujeito a observação do efeito da escrita no objeto (o que violaria a propriedade-ss) (SANDHU, 1993, AMOROSO, 1994). O cenário criado pela escrita cega torna-se uma preocupação pois o mesmo sujeito que é considerado inapropriado para verificar os conteúdos de um objeto tem permissão para fazer modificações arbitrárias nesse objeto, podendo causar problemas de integridade.

2.5.3 Modelos Baseados em Papéis (Role-Based Access Control Models)

As políticas **baseadas em papéis**, também chamadas de políticas RBAC (***Role Based Access Control***), regulam o acesso dos usuários aos recursos com base nas atividades que eles desempenham no sistema. Os papéis (*roles*) podem ser definidos como um conjunto de ações e responsabilidades associadas com uma atividade particular (SANDHU e SAMARATI, 1994). A atribuição de direitos de acesso é destinada a papéis (*roles*) e não a usuários, e os usuários recebem permissão para assumir e fazer parte de um ou mais papéis.

Um estudo do NIST (*National Institute of Standards and Technology*) (FERRAIOLO, GILBERT, *et al.*, 1993) demonstrou que o RBAC (*Role-Based Access Control*) trata muitas necessidades encontradas tanto no setor comecial quanto governamental. Nesse estudo foi identificado que muitas organizações baseiam suas decisões de controle de acesso "nos papéis que os usuários individuais desempenham na organização". As políticas baseadas em papéis vêm sendo utilizadas, na sua maioria, em aplicações comerciais e sistemas de bancos de dados (FERRAIOLO, GILBERT, *et al.*, 1993, RAMASWAMY e SANDHU, 1998, SANDHU, 1998) e vêm atraindo grande interesse da comunidade científica (KENT, 1999, ZURKO, SIMON, *et al.*, 1999, SANDHU, FERRAIOLO, *et al.*, 2000). *Roles* estão sendo considerados parte de padrões como SQL3 (***Structured Query Language*** 3) para gerência de bancos de dados; o uso administrativo de *roles* também é encontrado em sistemas operacionais como Novell Netware e Microsoft Windows NT (SANDHU, 1998). Alguns trabalhos acadêmicos usam as políticas baseadas em papéis em ambientes como o Java (NAGARATNAM, 1998, GIURI, 1998) e CORBA (BEZNOSOV e DENG, 1999).

Os modelos RBAC são **independentes de política**. Em (SANDHU, 1998), o autor considera que em um modelo RBAC, a política de autorização fica embutida nos vários componentes do RBAC. Esses componentes coletivamente determinam se um usuário particular tem direito de acesso a um item de dados particular.

Estes modelos a princípio devem suportar três princípios de segurança bem conhecidos (SANDHU, 1998): *least privilege* ou *princípio do privilégio mínimo* (apenas as permissões necessárias à execução de uma determinada tarefa são fornecidas aos membros de um papel), *separation of duties* ou separação de tarefas (fornece a garantia da existência de papéis mutuamente exclusivos na execução de determinadas atividades, tal como a existência de um *funcionário de contabilidade* e de um *gerente contábil* que participam na emissão de um cheque de uma empresa) e, *data abstraction* ou abstração de dados (é suportada pelas permissões abstratas como um débito ou crédito em um objeto contábil, ao invés de permissões como *read*, *write* e *execute* tipicamente fornecidas pelo sistema operacional)[8].

A idéia de *roles* é nova e antiga ao mesmo tempo. A falta de padronização de modelos RBAC provocou a existência de várias tentativas de definição das características de modelos RBAC (FERRAIOLO, BARKLEY, *et al.*, 1999, FERRAIOLO e KUHN, 1992, OSBORN, 1997, SANDHU, 1998) e também de vários tipos de implementações de políticas baseadas em RBAC. O próprio termo RBAC não tem um significado aceito em âmbito geral e é usado de forma diferente

[8] Entretanto, RBAC não pode garantir a aplicação desses princípios. O administrador da segurança pode configurar RBAC de forma que seja possível a violação desses princípios. Além disso, o grau de abstração de dados suportado será determinado por detalhes de implementação.

por usuários e implementadores. O modelo **RBAC-NIST** representa um passo em direção à padronização desta tecnologia (SANDHU, FERRAIOLO, *et al.*, 2000).

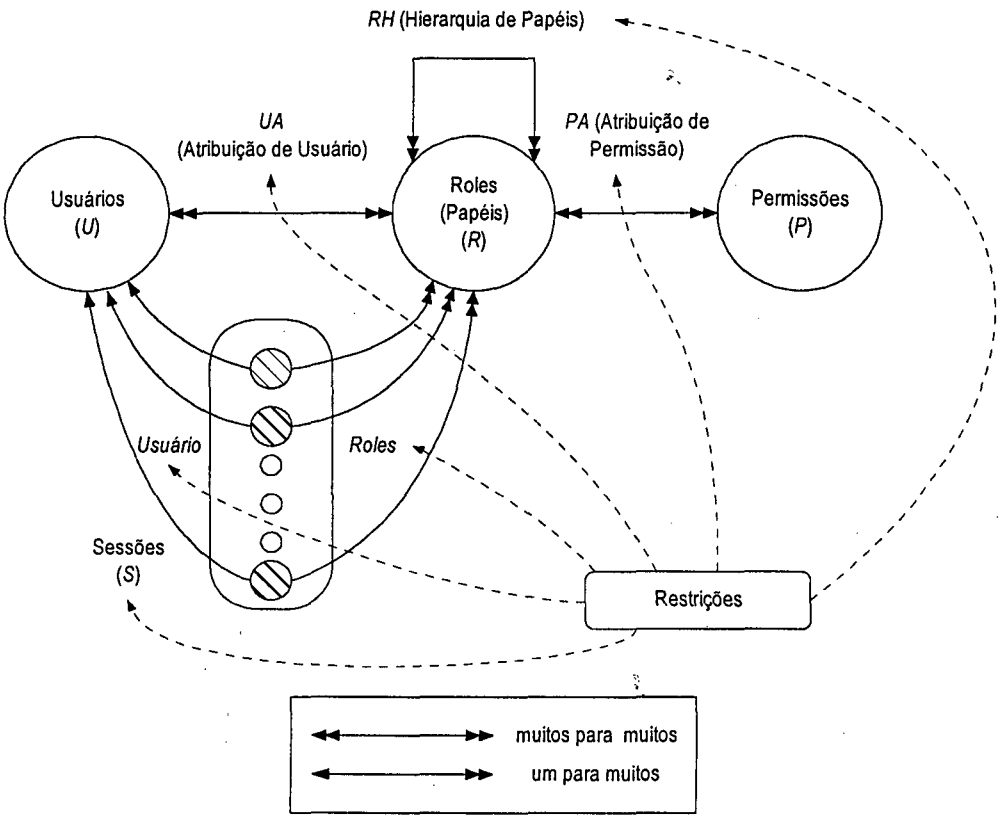


Figura 2.2 – Model Geral RBAC (SANDHU, COYNE, *et al.*, 1996).

O modelo **RBAC-NIST com restrições** (figura 2.2) representa um dos quatro modelos propostos na padronização de (SANDHU, FERRAIOLO, *et al.*, 2000). O modelo é baseado em três conjuntos de entidades chamadas usuários (*U*), papéis (*R*) e permissões (*P*). Intuitivamente, um **usuário** representa um ser humano ou uma entidade autônoma, um **papel** (*role*) é a designação de uma função dentro de uma organização que descreve a autoridade e a responsabilidade conferida a um membro desse papel e, uma **permissão** representa o direito a um acesso particular para um ou mais objetos no sistema.

As relações do RBAC exigem que a **associação usuário-papel** (UA, de *User Assignment*) e a **associação permissão-papel** (*PA*, de *Permission Assignment*), sejam relações de muitos para muitos (isto é indicado pelas setas cheias da figura 2.2). Um usuário pode pertencer a muitos papéis e um papel pode ter muitos usuários. De forma similar, um papel pode ter muitas permissões e uma permissão pode ser atribuída a muitos papéis. A posição dos papéis atua como uma parte intermediária entre usuários e permissões. Existe uma ordem parcial designada **hierarquia de**

papéis *RH* (***Role Hierarchy***), também representada como $\geq$, onde $x \geq y$ significa que o papel x herda permissões atribuídas ao papel y. Hierarquias são uma forma natural de estruturar papéis para refletir as linhas de autoridade e responsabilidade de uma organização. Por convenção, os papéis mais poderosos (chamados papéis *senior*) são representados no topo dos diagramas hierárquicos e os papéis menos poderosos (chamados papéis *junior*) são representados na parte inferior destes diagramas. Por exemplo, um papel designado *Supervisor de Projeto* em uma organização, pode herdar papéis de ambos funcionários *Engenheiro de teste* e *Programador*, que igualmente são membros de um projeto.

A Figura 2.2 mostra um conjunto de **sessões** (*sessions*). Cada sessão faz o mapeamento de um usuário a, possivelmente, muitos papéis. Intuitivamente, usuários estabelecem sessões durante as quais eles podem ativar um subconjunto de papéis aos quais pertencem (direta ou indiretamente através da hierarquia de papéis). A linha com seta dupla indo das Sessões (*S*) em direção aos Papéis (*R*) na Figura 2.2 indica que múltiplos papéis são ativados simultaneamente. As permissões disponíveis para o usuário são a união das permissões de todos os papéis ativos para aquela sessão. Cada sessão é associada com um único usuário, conforme indicado pela linha com seta única indo das Sessões (*S*) para os Usuários (*U*). Essa associação permanece constante pelo tempo de vida da sessão. Um usuário pode ter múltiplas sessões abertas ao mesmo tempo, cada uma delas em uma janela diferente da tela de uma estação de trabalho, por exemplo. O conceito de sessão se iguala à noção tradicional de *sujeito* da literatura de controle de acesso. Um sujeito (ou sessão) é a unidade de controle de acesso, e o usuário pode ter múltiplos sujeitos (ou sessões) com diferentes permissões ativas ao mesmo tempo.

Finalmente, a Figura 2.2 mostra uma coleção de **restrições** (*constraints*). Restrições se aplicam a quaisquer dos componentes precedentes e algumas vezes são consideradas a principal motivação para o uso de modelos RBAC. Um exemplo comum de restrições é o dos papéis mutuamente exclusivos, tais como *gerente de compras* e *gerente de pagamento de contas*. Em algumas organizações o mesmo indivíduo não pode ser membro dos dois papéis, pois isso criaria possibilidade do mesmo cometer fraudes. Isto concretiza a necessidade do princípio *separation of duties*.

2.5.4 Outros Modelos

Na literatura, além do modelo de Matriz de Acesso, Bell e Lapadula e modelos Baseados em Papéis, são identificados outros modelos. Mesmo que o modelo Bell e Lapadula tenha influenciado vários sistemas nas suas tentativas de implementar políticas obrigatórias, multinível, esse modelo vem sofrendo ao longo dos anos extensões ou adaptações. O modelo de Biba (BIBA,

1977, LANDWEHR, 1981) é uma versão do Bell e Lapadula para assegurar a integridade em sistemas com políticas multinível.

Modelos como o Clark-Wilson constituem um *framework* descritivo para expressar políticas de autorização de aplicações comerciais (CLARK e WILSON, 1987). O objetivo desse modelo é lidar com a integridade de dados, prevenindo modificação não autorizada de dados, fraudes e erros (MAYFIELD, ROSKOS, *et al.*, 1991).

Modelos de controle de fluxo são introduzidos no sentido de tratar os problemas de confinamento (canais cobertos temporais, canais de memória, cavalos de tróia etc). Não se considera mais nesses modelos os acessos de escrita e de leitura sobre arquivos, por exemplo, mas dos fluxos de informação entre sujeitos. Esses modelos tratam de identificar os canais legítimos de comunicação e os ilegítimos. O modelo de (DENNING, 1977) é um exemplo dessa classe de modelos. Existem também outros modelos que tratam da inferência de informações em bases de dados. O modelo de não-inferência apresentado em (GOGUEM e MESEGUER, 1984) é um exemplo desses modelos.

2.5.5 Mecanismos de Segurança

Os **mecanismos de segurança** em um sistema concretizam as políticas de autorização e de autenticação definidas para o mesmo. Esses mecanismos asseguram que todos os acessos a objetos no sistema são autorizados pela política de autorização definida.

O **controle de acesso** é a mediação das requisições de acesso a objetos feitas pelos sujeitos, e é executado pelo **monitor de referência** do sistema, responsável por permitir ou negar o acesso correspondente. O monitor de referência é o modelo conceitual e a entidade abstrata responsável pelo controle de acesso definido por (LAMPSON, 1971).

Os monitores de referência intervêm nos vários níveis de um sistema. As referências a segmentos de memória são validadas nas camadas inferiores do sistema, por meio do *hardware* e do sistema operacional; por sua vez o serviço de diretório valida os acessos a arquivos.

A noção de **núcleo de segurança** é definida em (LANDWEHR, 1983) como um conjunto de recursos de *hardware* e *software* implementando o conceito de monitor de referência. O núcleo de segurança deve obrigatoriamente possuir três propriedades: deve ser inviolável; deve sempre ser invocado e deve ser verificável quanto à correção.

A primeira propriedade corresponde a assegurar o isolamento do monitor: não deve ser possível modificar o comportamento do monitor quando o sistema estiver em funcionamento. A segunda propriedade indica que o acesso deve ser inevitável e obrigatoriamente verificado pelo monitor de referência. A terceira propriedade determina que o monitor de referência deve ser simples o suficiente para ser analisado e verificado: devemos nos assegurar que o monitor faz

corretamente o que supomos que ele deva fazer. A Figura 2.3 esquematiza o papel do monitor de referências na validação dos acessos.

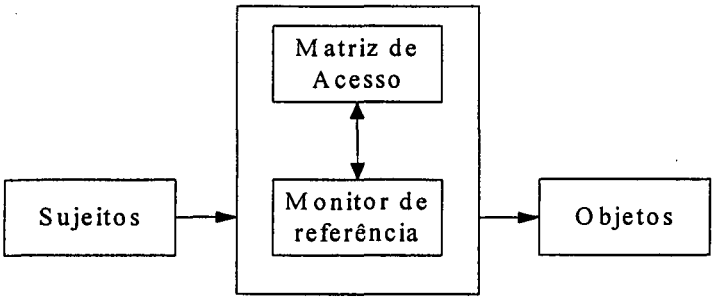


Figura 2.3 – Monitor de referência.

2.5.5.1 Base Computacional de Confiança (TCB – Trusted Computing Base)

A **base computacional de confiança ou TCB** (*TCB - Trusted Computing Base*), introduzida pelo livro laranja ou TCSEC[9] (*Trusted Computer System Evaluation Criteria* (DOD, 1985)), corresponde à totalidade dos mecanismos de segurança no interior do sistema, incluindo *hardware*, *software* e *firmware*, cuja combinação é responsável pela concretização de uma política de autorização. A habilidade do TCB em garantir a aplicação da política de autorização de forma correta depende somente dos mecanismos presentes no TCB e, da entrada correta de parâmetros relacionados com a política de autorização, por exemplo, o nível de habilitação do usuário.

Dessa forma, podemos dizer que o monitor de referência é um conceito abstrato, o núcleo de segurança é a implementação desse conceito, e o TCB contêm o núcleo de segurança e os controles adicionais, como os controles criptográficos, serviços de autenticação, serviços de identificação, etc, que implementam as políticas de segurança do sistema.

A **criptografia** é a ciência e arte da escrita secreta e do armazenamento secreto de informações, e possui uma grande importância na implementação de todos os serviços de segurança. Além de fornecer a confidencialidade, as técnicas criptográficas servem de suporte a serviços como: autenticação, integridade e não-repudiação.

[9] O TCSEC é uma classificação de sistemas informáticos seguros proposta pelo DoD (*Department of Defense* – Departamento de Defesa dos Estados Unidos), fundamentado em avaliações qualitativas. Essa classificação é baseada na presença ou ausência de determinados mecanismos, na validação do sistema em relação a um modelo etc. No TCSEC são apresentados critérios para uma classificação em sete níveis de sistemas seguros. A avaliação de um produto consiste a atribuir ao mesmo um dos sete níveis.

Vários elementos fazem parte de um sistema de cifragem : o algoritmo de cifragem, as chaves utilizadas, o comprimento da chave, o texto claro o e o texto cifrado. O algoritmo é a função matemática que executa a cifragem e decifragem dos textos. A chave é um parâmetro do algoritmo que determina como seus dados são transformados. Os algoritmos de cifragem e decifragem são usados para transformar, a partir de uma chave, o texto claro (texto em claro) em um texto cifrado e vice-versa. Normalmente, a segurança desses algoritmos de cifragem e decifragem depende da segurança das chaves. Isso significa que o algoritmo pode ser publicado e analisado, ainda que existam ainda algoritmos não públicos.

Existem dois tipos gerais de algoritmos criptográficos: **algoritmos simétricos** e **algoritmos assimétricos**. Os algoritmos simétricos usam a mesma chave para cifrar e decifrar a mensagem. Também são conhecidos como algoritmos de chave secreta. Os algoritmos assimétricos usam uma chave pública para cifrar a mensagem e uma chave privada para decifrar a mensagem. Os sistemas assimétricos também são conhecidos como algoritmos de chave pública.

Enquanto nos algoritmos de chave pública a segurança dos mesmos está centrada na complexidade computacional, a força dos algoritmos simétricos normalmente se baseia na "confusão estatística" conseguida com permutações e substituições sucessivas executadas nos dados.

Exemplos de sistemas de chave secreta são: DES (*Data Encryption Standard*) (SMID e BRANSTAD, 1988), que usa chave de 56 bits, RC5 (RIVEST, 1995), que usa chaves de tamanho definido pelo usuário, e IDEA (LAI, 1992), que usa chave de 128 bits. Sistemas de chave pública que podem ser citados como exemplo são RSA (Rivest Shamir Adelman) (RIVEST, SHAMIR, *et al.*, 1978) e ElGamal (ELGAMAL, 1985).

A **autenticação** relaciona-se a um cenário onde algum requisitante apresenta uma identidade de principal e diz ser este principal. A autenticação habilita o verificador a ter uma certa garantia de que indivíduos, emissores de mensagens e mesmo conteúdos de mensagens, em sistemas distribuídos, são legítimos e íntegros, conforme o contexto que se esteja tratando.

Os métodos de autenticação são baseados em qualquer um dos seguintes princípios: o requisitante demonstra conhecimento de alguma coisa, por exemplo, de uma senha; o requisitante demonstra ter alguma coisa, por exemplo, uma chave ou um cartão físico; o requisitante exibe alguma característica imutável, por exemplo, uma impressão digital; é apresentada uma evidência de que o requisitante está em algum lugar particular em um certo instante, ou ainda, o verificador aceita que alguma parte interessada, que é confiável, já tenha estabelecido a autenticação.

Uma das formas mais simples de autenticação usada em sistemas distribuídos são as senhas associadas ao processo de *login* dos principais, que garantem que a identidade do usuário está correta.

Outros mecanismos de autenticação para sistemas distribuídos, confiam no uso de controles criptográficos para garantir a segurança. Esses controles criptográficos se baseiam na existência de um **serviço de distribuição de chaves** - que representa os meios seguros para gerar, armazenar e distribuir as chaves criptográficas necessárias.

Dois mecanismos de autenticação baseados em criptografia são o **Kerberos** (baseado em chave secreta) (COULOURIS, 1994) e o **X.509** (ITU-TX509, 1993, GARFINKEL e SPAFFORD, 1997) (baseado em chave pública).

Atualmente, em ambientes de programação distribuída como a Internet, podem existir níveis de autenticação diferentes na execução de uma aplicação como: a autenticação do código móvel, autenticação do principal e autenticação da sessão estabelecida entre as partes comunicantes.

Embora o TCSEC afirme que o núcleo de segurança e o TCB são essenciais para a construção de sistemas seguros, existe atualmente uma discussão sobre a aceitação desta base confiável. Alguns autores consideram inadequada a idéia de entidades "confiáveis" em sistemas de informação modernos (BLAKLEY, 1997). Neste sentido, as questões normalmente levantadas sobre o TCB são as seguintes:

- Sob o ponto de vista de quem o TCB pode ser considerado confiável ? A partir de que funções concretizadas o mesmo pode ser considerado confiável ?
- Qual a forma de definir os limites do TCB, ou seja, como saber exatamente quais os elementos do sistema que devem ficar dentro e fora do TCB ?

É interessante observar também a equivalência de nomenclatura adotada pelos Critérios Comuns[10] (*CC - Common Criteria*) para Avaliação de Segurança, padrão ISO 15408 de 1999 (ISO/IEC 15408-1, 1999, ISO/IEC 15408-2, 1999, ISO/IEC 15408-3, 1999), para representar esta base confiável no sistema. O CC é uma evolução do TCSEC. Nos Critérios Comuns, ao invés do conceito de núcleo de segurança, existe o conceito de **mecanismo de validação de referência** (*Reference Validation Mechanism*), com o mesmo sentido definido no TCSEC. Nos Critérios Comuns, ao invés do conceito de TCB, existe o conceito de **funções de segurança do sistema**

[10] Os Critérios Comuns (*Common Criteria – CC*) para avaliação da segurança são o resultado de um esforço de várias organizações internacionais para desenvolver um único padrão de avaliação da segurança para sistemas e produtos de tecnologias de informação distribuídos (http://www.commoncriteria.org). Esse critério deriva de padrões anteriores como o TCSEC (ou livro laranja que data dos anos 80), o ITSEC (critério europeu de 1991), o CTCPEC (critério canadense de 1993) e o FC (união dos critérios europeu e canadense de 1993). As organizações colaboradoras que participam destes esforços são: CSE (Canadá), SCSSI (França), BSI (Alemanha), NLNCSA (Holanda), CESG (Inglaterra) e NIST e NSA (Estados Unidos).

(*TSF – Target of Evaluation Security Functions*), que corresponde ao conjunto de todo o *hardware*, *software* e *firmware* do sistema, necessariamente confiável, para que se tenha a concretização correta das políticas de segurança no sistema.

2.6 Discussão sobre a Implementação de Mecanismos de Segurança em Sistemas Distribuídos

A segurança em um sistema distribuído é uma tarefa certamente muito complexa de se realizar. Se já é difícil assegurar de maneira independente a segurança de cada computador hospedeiro em uma rede isoladamente, é uma tarefa muito mais difícil garantir a segurança do sistema global distribuído. Fatores como a **escalabilidade** (*scalability*) dos sistemas distribuídos impõem maiores níveis de complexidade na **autenticação**, na **autorização** e na **nomeação dos recursos** em sistemas distribuídos (NEUMAN, 1994).

Existem algumas abordagens no sentido da implementação de mecanismos de segurança nesses sistemas.

2.6.1 Abordagem centralizada

A implementação de mecanismos de segurança baseada em monitor de referência e TCB parece impor intuitivamente uma gestão centralizada. Em sistemas distribuídos isso consiste em colocar toda a confiança e responsabilidade da segurança do sistema em uma máquina. Essa máquina, de modo geral, desempenha o papel de um monitor de referência: essa máquina será o mediador não contornável a todas interações entre sujeitos e objetos autorizadas no sistema.

A vantagem dessa abordagem é que ela permite se ter uma política de autorização coerente e fácil de se montar: a política de autorização é gerida por uma só máquina. Essa abordagem, além dos aspectos de tolerância a falhas e de desempenho, é facilmente criticável pela concentração de funções e informações sensíveis em uma só máquina. Essa máquina seria alvo privilegiado para eventuais intrusões no sistema.

2.6.2 A abordagem do TCSEC

Uma outra abordagem para sistemas distribuídos é a apresentada no TCSEC (DOD, 1985, NICOMETTE, 1996) onde cada sítio do sistema possui uma TCB, encarregada de controlar os acessos locais e acessos remotos. Um sujeito (na Figura 2.4, representado pela letra S) que deseja acessar um objeto remoto (representado pela letra O na Figura 2.4) repassa seu pedido de acesso ao

TCB local, que, por sua vez, contacta o TCB remoto que valida ou recusa o acesso ao objeto sob seu controle.

Nessa abordagem a segurança do sistema depende da confiança mútua dos TCBs em relação aos seus pares. Esse conjunto de TCBs que possuem confiança mútua é chamado de *Network Trusted Computing Base* (NTCB). A segurança global do sistema repousa na segurança do NTCB (ver Figura 2.4).

A gestão descentralizada das informações pode provocar alguns problemas de coerência. A matriz de acesso se apresenta nesse caso particionada envolvendo todos os sítios da rede, o que pode dificultar a gestão de uma política de autorização coerente. Uma das fraquezas desse esquema é a confiança que se tem que assumir em relação a todos os administradores de cada sítio. Se um administrador é corrompido, o sistema global é corrompido pois as outras TCB's são construídas na base da confiança mútua.

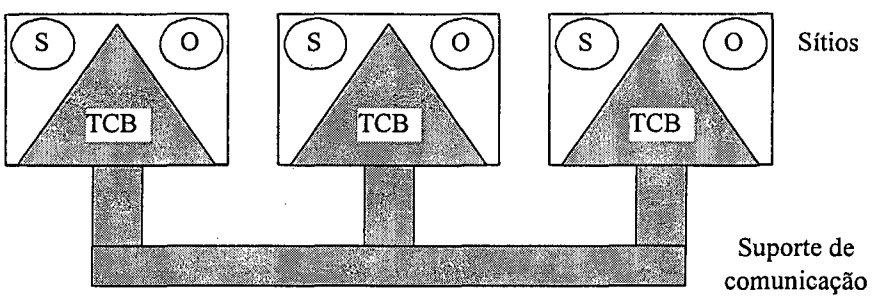


Figura 2.4 – Abordagem NTCB.

2.6.3 Abordagens baseadas no particionamento de funções de segurança no sistema

2.6.3.1 Abordagem Kerberos

Essa abordagem combina aspectos das duas abordagens anteriores visando o particionamento de funções de segurança sobre sítios confiáveis. Essa é a abordagem usada no projeto Athena, desenvolvido no MIT, e de muitos outros sistemas presentes na literatura. Assim, no projeto Athena, a autenticação do sistema é gerida por um servidor único (o Kerberos (NEUMAN e TS'O, 1994)), numa abordagem centralizada, enquanto a autorização é controlada independentemente por cada sítio do sistema. O servidor Kerberos deve ser confiável. Todo o cliente querendo acessar informações em servidores deve obrigatoriamente se fazer autenticar pelo servidor de autenticação, podendo na seqüência acessar as informações nos servidores segundo a autorização possuída.

Nessa abordagem, ao contrário da anterior, os servidores não necessitam confiar no sítio do cliente. Porém, os mesmos aspectos levantados nas abordagens anteriores podem ser verificados na abordagem desse item. Ou seja, a gestão distribuída implica na dificuldade de manter a coerência na gestão da política de autorização do sistema (vários servidores decidindo concorrentemente) e o servidor único de autenticação é um ponto vulnerável do sistema. A Figura 2.5 sintetiza essa abordagem. Mais detalhes sobre os protocolos de autenticação do Kerberos podem ser encontrados na literatura (COULOURIS, 1994, MILLER, NEUMAN, *et al.*, 1988, NEUMAN e TS'O, 1994).

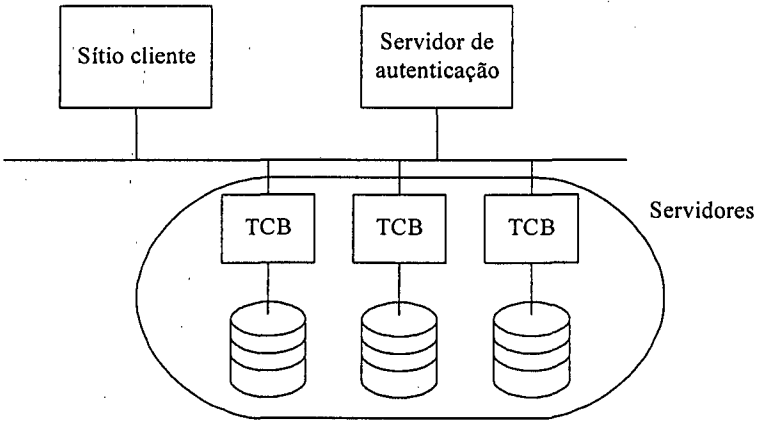


Figura 2.5 – Abordagem Kerberos.

2.6.3.2 Abordagem Delta-4

Essa abordagem não é fundamentada na existência de um sítio único confiável. Na verdade é definido um *quorum* de sítios de autenticação. Nessa abordagem, um cliente tem acesso a objetos em servidores desde que consiga a autenticação em uma maioria dos servidores de autenticação. Essa abordagem tolera falhas ou instrusões em uma minoria desses servidores de autenticação. Para isso essa abordagem usa nos servidores de autenticação a técnica de fragmentação e disseminação (DESWARTE, 1991, FRAGA, 1985).

Na verdade, essa abordagem da fragmentação e disseminação se apoia sobre a noção da repartição geográfica para melhorar a segurança: a intrusão de uma minoria dos servidores de autenticação dá acesso só a fragmentos isolados de informações sensíveis do serviço de autenticação (Figura 2.6).

Nessa abordagem o servidor de segurança é composto de vários sítios de segurança, todos com as mesmas funcionalidades, mas não com os mesmos dados. Os sítios clientes e os outros servidores não fazem distinção entre os sítios de segurança. Cada sítio de segurança é administrado por um administrador diferente. O administrador de um sítio particular não tem nenhum direito sobre os outros sítios de segurança. Esta separação de poderes evita o problema clássico de sistemas distribuídos onde um administrador único tem todo o poder sobre as máquinas do sistema.

É importante salientar que nessa abordagem os sítios de segurança são responsáveis pela autenticação no sistema distribuído mas também pela gestão dos acessos a objetos persistentes no sistema. A autorização para efetuar uma operação no sistema é validada pelo servidor de segurança.

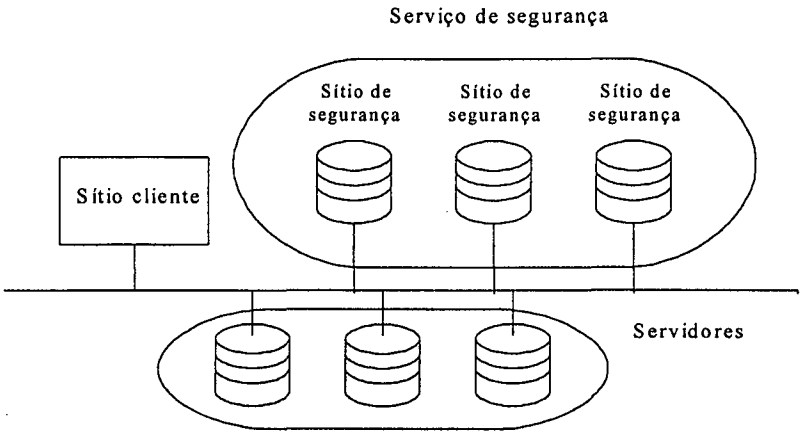


Figura 2.6 – Abordagem Delta-4.

2.6.3.3 Abordagem X.509

Nessa abordagem introduzida como um padrão ITU-T (ITU-TX509, 1993), o serviço de autenticação é particionado no sentido de aumentar a sua aplicabilidade em redes de larga escala como a Internet (GARFINKEL e SPAFFORD, 1997). Diferentes autoridades certificadoras (CA – *Certification Authority*) são estruturadas na forma de uma árvore. Um cliente particular é registrado em um desses CAs. O acesso de um cliente a objetos de um servidor registrado em outro CA implica na interação entre os CAs do cliente e do servidor no processo de autenticação. Mais detalhes sobre o protocolo X.509 são descritos em (ITU-TX509, 1993).

2.6.3.4 O Protocolo LDAP

O padrão X.500 organiza entradas de diretórios num espaço de nomes hierárquico. Este padrão também define o protocolo de acesso ao diretório *DAP* (***Directory Access Protocol***). O DAP necessita da pilha de protocolos OSI para operar, recurso esse nem sempre disponível. Assim, é desejável uma interface de 'peso' menor (*Lightweight*) para o serviço de diretório (JOHNER, BROWN, *et al.*, 1998, HASSLER, 1999, SANTIN, 2000, WAHL, HOWES, *et al.*, 1997).

O LDAP (***Lightweight Directory Access Protocol***) é uma implementação do serviço de diretórios baseado no X.500, mas implementado com a pilha de protocolos TCP/IP, que possui algumas simplificações de operações e até omissões de características do X.500.

O modelo de nomes do LDAP define como as entradas do diretório são organizadas e identificadas. As entradas são organizadas numa estrutura similar a de árvore dos sistemas de arquivos denominada DIT (*Directory Information Tree*). Cada entrada é identificada através de um DN (*Distinguished Name*), que é formado por um conjunto de atributos. Esses atributos podem ser de vários tipos, como por exemplo: nome comum (*CommonName* – cn), nome da organização (*OrganizationName* – o), nome do departamento (*OrganizationalUnitName* – ou), nome do país (*CountryName* – c) e outros. Um exemplo de DN pode ser: (cn=Carla M. Westphall, o=UFSC, c=BRASIL).

Um servidor LDAP pode não armazenar toda a DIT. A DIT pode ser dividida em partes e ficar fisicamente armazenada em locais diferentes. Logicamente, o servidor LDAP forma a DIT global usando o mecanismo de sufixos e *referrals*[11] (WAHL, HOWES, *et al.*, 1997). O sufixo é o nível hierárquico mais alto da DIT que o servidor armazena. O *referral* é a referência da outra parte da árvore, para a qual pode ser encaminhada uma pesquisa, caso seja constatado que o DN a ser pesquisado não consta no servidor local. A Figura 2.7 mostra um exemplo de DIT particionada e distribuída em dois servidores.

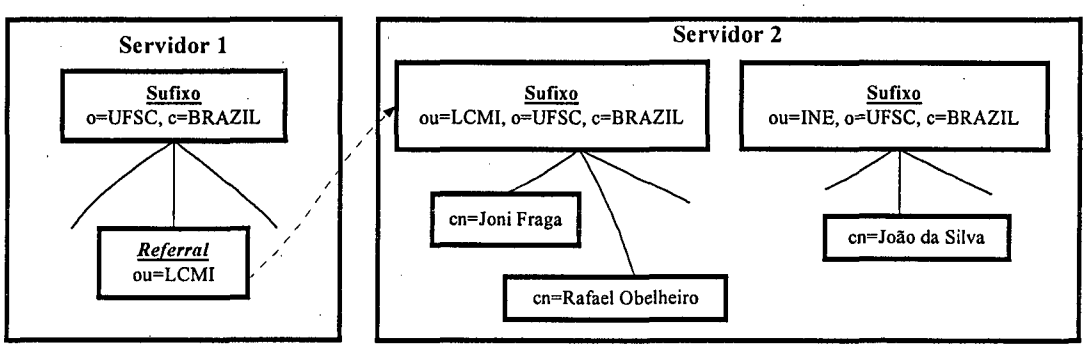


Figura 2.7 – Exemplo de DIT formada por dois servidores.

Quando uma pesquisa é feita, um cliente pode ter como resposta um *referral*. A API (*Application Programming Interface*) do LDAP permite ao programador especificar se este *referral* deve ser seguido automaticamente ou se deve ser retornado ao cliente para que ele decida se a segue ou não.

O LDAP tem formas de autenticar os usuários que desejam acesso às informações contidas no diretório, usando o *framework* para adicionar mecanismos de autenticação a protocolos orientados a conexão, denominado SASL (*Simple Authentication and Security Layer*). Com este framework, é possível realizar a autenticação usando os controles criptográficos como o SSL (FREIER, KARLTON, *et al.*, 1996). Dessa forma, o LDAP é uma alternativa bastante usada para

[11] *Referrals* existem a partir da versão 3 do LDAP.

implementar a abordagem de autenticação X.509 que se baseia no uso dos certificados X.509v3 (JOHNER, BROWN, *et al.*, 1998).

A *IETF* (*Internet Engineering Task Force*), ampla comunidade internacional preocupada com o desenvolvimento e evolução da Internet (http://www.ietf.org), está utilizando o LDAP para implementar estruturas de chaves públicas baseadas na abordagem X.509 na Internet (BOEYEN, HOWES, *et al.*, 1999). O serviço de nomes disponível para o ambiente Java, implementado pela Sun e chamado *JNDI* (*Java Naming Directory Interface*), também possui interfaces para uso de servidores LDAP. Mais detalhes sobre JNDI e LDAP podem ser obtidos em (http://java.sun.com/products/jndi/tutorial/ldap/index.html). Existem idéias também que consideram o armazenamento de objetos CORBA usando o diretório LDAP (RYAN e SELIGMAN, 1999).

2.6.4 Considerações sobre as abordagens a Implementação de Mecanismos de Segurança em Sistemas Distribuídos

Em sistemas distribuídos, no sentido de aproveitar as características desejáveis nesses sistemas de disponibilidade de serviços e de transparência da distribuição, a segurança é conduzida no sentido de algumas funções globais e outras locais. As abordagens baseadas no particionamento (item 2.6.3), preenchem essa visão de segurança. Na verdade, em sistemas distribuídos existem objetos persistentes de grande granularidade (arquivos, *daemons* sobre o Unix etc) e objetos criados dinamicamente no sentido de preencher tarefas pontuais. Uma maneira adequada de implementar um esquema de autorização é localizar o nível global, tratando os acessos aos objetos persistentes e também a autenticação, em um único servidor de autorização e autenticação. O nível correspondendo a segurança local, envolvendo o controle sobre objetos temporários ou localizados, é tratado pelos núcleos de segurança e TCB's locais. O sistema Delta-4 (DESWARTE, 1991, FRAGA, 1985) se utiliza dessa abordagem.

A definição de um *quorum* de servidores de autorização minimiza as necessidades de entidades confiáveis no sistema. A colocação de funções de controle de acesso nesses servidores de autorização e autenticação, definindo um nível global no esquema de autorização e autenticação, minimiza também o impacto na violação de um núcleo de segurança ou uma máquina do sistema.

A escalabilidade (*scalability*) dos sistemas distribuídos impõem maiores níveis de complexidade na autenticação, na autorização e na nomeação dos recursos nestes sistemas. Em redes de larga escala, como a Internet, fica difícil um controle de acesso a objetos persistentes em um nível global. As funções de nível global nesses sistemas se limitam a um servidor de autenticação normalmente na forma hierarquizada (X.509 e LDAP).

2.7 Conclusões do Capítulo

Nesse capítulo foram apresentados os principais conceitos relacionados com a segurança (*security*) em sistemas informáticos. Foram discutidas as ameaças mais comuns presentes nos sistemas e a necessidade do estabelecimento de políticas de autorização para conter tais ameaças. Os modelos de segurança apresentados são importantes para o entendimento da dinâmica associada com as políticas correspondentes. Foi feita uma discussão sobre a implementação dos esquemas de autorização em sistemas distribuídos, considerando as principais tendências atuais. O estudo descrito nesse capítulo não teve a pretensão de ser exaustivo no assunto, mas colocar as abordagens que podem ser adotadas na condução do tratamento da segurança em sistemas distribuídos.

Capítulo 3

SEGURANÇA EM SISTEMAS ABERTOS DISTRIBUÍDOS:

A PROPOSTA *JACOWEB*

3.1 Introdução

Os paradigmas e ferramentas de programação distribuída representados pelo Java, CORBA e Web concretizam o conceito de objetos distribuídos em sistemas de larga escala como a Internet (GHEZZI e VIGNA, 1998, RESNICK, 1996). Cada uma destas ferramentas introduz suas próprias estruturas e conceitos com relação a segurança.

O **Projeto *JaCoWeb Security* (*http://www.lcmi.ufsc.br/jacoweb*)** tem como objetivo o uso das especificações CORBA de segurança integrada com os suportes de segurança Java e Web de modo a compor um esquema de autorização para aplicações distribuídas em redes de larga escala. Esse esquema está sendo desenvolvido com o objetivo de concretizar a implantação de políticas globais, o que representa um grande desafio, principalmente em redes de larga escala como a Internet (WESTPHALL e FRAGA, 1999a, 1999b, WESTPHALL, FRAGA, *et al.*, 2000). Nestes ambientes, fazendo uso do esquema *JaCoWeb*, aplicações distribuídas são expressas na forma de clientes representados por *applets* Java, e de objetos servidores de aplicação, visíveis via CORBA.

Este capítulo apresenta os suportes de segurança das plataformas Java e Web de forma sucinta e também descreve as especificações CORBA de segurança. O esquema de autorização *JaCoWeb* é também introduzido neste capítulo, através de suas principais funcionalidades definidas a partir da integração das especificações CORBA e de suportes das ferramentas Java e Web.

Nos capítulos subseqüentes são explorados os aspectos de implementação de modelos de segurança como o Matriz de Acesso, o Bell e Lapadula e modelos Baseados em Papéis no esquema de autorização *JaCoWeb*. Estes modelos de segurança são adaptados no esquema *JaCoWeb* no sentido de oferecer soluções para a implementação de políticas de segurança em aplicações distribuídas em ambientes de larga escala. Considerações sobre o *JaCoWeb* em ambientes de larga escala, bem como comentários sobre trabalhos e pesquisas relacionadas, também são apresentados.

3.2 JAVA E WEB

A linguagem **Java** popularizou o conceito de **código móvel**[12] através dos seus ***applets*** executados a partir de *browsers* Web (GOLLMANN, 1999, THORN, 1997). O ambiente Web caracteriza então a estrutura mais simples para a execução de códigos móveis, possibilitando a carga destes códigos em qualquer ponto da rede mundial.

Java foi projetada para executar ***applets*** e **aplicações, dinâmicas** e altamente **interativas** em uma rede de computadores. O suporte necessário para a execução de códigos Java torna esses códigos totalmente independentes dos sistemas operacionais e máquinas disponíveis em uma rede.

O ambiente de desenvolvimento Java compreende **três componentes principais** (MCGRAW e FELTEN, 1997, SUN, 1995a, 1995b): uma linguagem de programação que produz como resultado da compilação um código intermediário e independente de arquitetura chamado ***bytecode***; a **máquina virtual Java** (JVM – *Java Virtual Machine*) que interpreta o *bytecode* e atua como um sistema operacional portável para executar objetos; um **conjunto de classes ou APIs** que são executadas sobre o interpretador (JVM) e fornecem algumas classes básicas, úteis na construção de aplicações. A grande vantagem da interpretação é que a mesma permite que aplicações Java sejam executadas em qualquer sistema que possua uma implementação da máquina virtual Java.

O ambiente Java fornece uma nova forma de desenvolver, gerenciar e executar aplicações. Pode-se acessar a última versão de uma aplicação de forma automática; pode-se distribuir aplicações para milhões de clientes disponibilizando-a no servidor Web sem a preocupação com instalação e atualização de *software*. Java também é muito boa para implementar servidores, permitindo que seus serviços sejam movidos para uma máquina hospedeira onde os mesmos se fazem necessários (EVANS e ROGERS, 1997).

Java é uma linguagem que pode ser usada na construção de objetos distribuídos suportados pelo CORBA. A linguagem Java permite executar objetos CORBA em vários tipos de ambientes, simplificando a distribuição de código em grandes sistemas CORBA. Java é uma ótima linguagem para escrever objetos clientes e servidores CORBA. Suas características de *multithreading*, coletor de lixo automático e gerência de erros facilita a escrita de objetos de rede mais robustos.

[12] O *código móvel*, neste trabalho, se refere ao *software* que viaja através de uma rede heterogênea, atravessando domínios de proteção e sendo executado automaticamente no seu destino.

3.2.1 Modelo de segurança Java

A **segurança** representa um grande problema nos sistemas que fornecem suporte à mobilidade de código e freqüentemente é considerada **a principal limitação para o amplo uso desses paradigmas** (THORN, 1997). Existem três categorias básicas de ataques potenciais que os *applets* Java poderiam facilitar: ataques à integridade do sistema, através da remoção ou modificação de arquivos, da modificação da memória atual em uso, do término de processos ou *threads*; ataques à disponibilidade do sistema, através da alocação de, por exemplo, grandes quantidades de memória e pela criação de *threads* de alta prioridade negando o uso legítimo da máquina pela retenção de recursos; e ataques que invadem a privacidade do usuário (MCGRAW e FELTEN, 1997, SUN, 1997b).

O modelo de segurança implementado pela plataforma Java, na sua proposição inicial, é centrada sobre o conceito de ***sandbox*** (SUN, 1998). A essência do modelo *sandbox* é que o código local é confiável e tem acesso completo aos recursos do sistema (como o sistema de arquivos) enquanto o código remoto (um *applet*) não é confiável e pode acessar apenas recursos limitados, fornecidos dentro do *sandbox*. Esse conceito de *sandbox* é empregado pelo *toolkit* de desenvolvimento Java (*Java Development Kit* – JDK) e é geralmente adotado pelas aplicações construídas a partir do JDK, incluindo os *browsers* Web habilitados a executar códigos Java.

A segurança na plataforma Java é **concretizada em vários níveis**. Primeiramente, a linguagem, que é interpretada, é projetada para ser segura em relação aos tipos de dados utilizados. Outra parte importante desse modelo de segurança está nas **ações do compilador** e do verificador de *bytecodes* que garantem apenas a execução de códigos Java legítimos. Ao ser compilado, o código já sofre uma verificação de tipos minuciosa. Porém, como os *browsers* que carregam os arquivos de classes não sabem se os *bytecodes* correspondentes foram produzidos por um compilador Java confiável, um **verificador de *bytecodes*** (*bytecode verifier*) é ativado para inspecionar esse código antes de sua execução sobre a máquina virtual (*Java Virtual Machine* - JVM). O verificador de *bytecodes* representa um mecanismo de controle de acesso estático baseado na inspeção de código.

Também uma peça importante na segurança Java é o **carregador de classes** (*class loader*), uma ferramenta programável que fornece os meios para recuperar e ligar, de forma dinâmica, as classes de uma aplicação em uma JVM, fornecendo assim a idéia de mobilidade da linguagem. O *class loader* é chamado pela JVM em execução quando o código que está executando contém uma referência não resolvida a um nome de uma classe. O *class loader* recupera a classe correspondente, possivelmente de um *host* remoto e então carrega a classe na JVM. Nesse ponto o código correspondente é verificado e executado na máquina virtual (JVM).

Ao ser executado, o acesso aos recursos do sistema é mediado pela JVM, através de uma classe chamada *Security Manager* (**gerente de segurança**), que restringe as ações de um código não confiável ao mínimo possível. Esse mecanismo implementa o controle de acesso (dinâmico) na execução dos códigos, concretizando a idéia do *sandbox*.

Duas fases de evolução podem ser citadas a partir do modelo de segurança *sandbox* original (SUN, 1998). A primeira envolve o *kit* JDK 1.1.x que introduziu o conceito de ***applet* assinado**. Nesse modelo, um *applet* assinado digitalmente é tratado como se fosse um código local correto, desde que a chave da assinatura seja reconhecida como confiável pelo sistema que recebeu o *applet* para execução. Além do conceito de *applet* assinado, JDK 1.1.x definiu uma nova API, chamada de *Java Cryptography Architecture API*, que foi projetada para fornecer aos desenvolvedores de aplicações o acesso a funcionalidades criptográficas na plataforma Java (SUN, 1999). No *kit* JDK 1.1, essa "arquitetura de funções criptográficas" incluiu classes e interfaces para prover assinaturas digitais e *message digests*.

A segunda evolução do modelo de segurança do Java é apresentada no *kit* JDK 1.2 onde estão presentes os seguintes objetivos: prover **controle de acesso de granularidade fina**, possibilitar **políticas de segurança facilmente configuráveis**, definir uma estrutura de controle de acesso que pode ser **estendida** facilmente para todos os programas Java, incluindo aplicações e *applets*. O conceito de **domínios de proteção** é fundamental nessa nova arquitetura, cumprindo em parte esses objetivos citados. Um domínio pode ser definido como o conjunto de objetos que são diretamente acessados por um principal. Domínios de proteção possuem duas categorias distintas: domínios do sistema e domínios da aplicação. É importante que todos os acessos aos recursos externos, tais como sistema de arquivos, serviços de rede, tela e teclado sejam acessíveis apenas através dos domínios do sistema.

Uma política de segurança do sistema, definida pelo usuário ou pelo administrador do sistema, deve especificar quais domínios de proteção devem ser criados e quais permissões devem ser fornecidas nesses domínios. O ambiente de execução Java mantém um mapeamento do código (classes e instâncias) para seus domínios de proteção e permissões correspondentes. O conceito de domínio implementa de certa forma o princípio do privilégio mínimo (*least privilege*).

No JDK 1.2, é também feita uma extensão à API criptográfica no sentido de suportar certificados X.509v3 (SUN, 1999).

3.2.2 Modelo de Segurança Web

O modelo de programação distribuída fornecido a partir do ambiente WWW é extremamente poderoso, escalável e útil no desenvolvimento de aplicações distribuídas. A existência de um ambiente desse porte que seja seguro é necessário quando se têm objetivos de

utilizá-lo nas mais diversas aplicações em um contexto de redes de larga escala (RUBIN, GEER, *et al.*, 1997).

A evolução da Web tem acrescentado novas características a esse ambiente para satisfazer a crescente demanda, sem considerar de forma cuidadosa, pelo menos inicialmente, o impacto na segurança do sistema. De uma maneira geral, o problema de **segurança da Web** pode ser dividido em três partes: segurança do servidor, segurança da informação em trânsito e segurança do cliente.

A **segurança do servidor** está fundamentada em serviços de autenticação e de controle de acesso. Os serviços de autenticação disponíveis são o esquema básico, o esquema com *digest authentication* e o esquema com certificados (RUBIN, GEER, *et al.*, 1997). O esquema básico de autenticação é um sistema de identificação baseado no fornecimento de um nome de usuário e de uma senha. Não existe no mesmo uma preocupação em fornecer alguma forma de controle que dê garantias da validade dessas informações. O esquema designado *digest authentication* fornece alguns mecanismos de troca permitindo autenticação dessas informações por meio de um *digest* calculado sobre essas informações de identificação. Na autenticação em sistemas distribuídos de larga escala, é desejável a intermediação entre os pares comunicantes de uma entidade confiável (autoridade de certificação) que, através de emissão e revogação de certificados, garanta as trocas de informações na autenticação de um cliente perante um servidor.

Existem duas formas de controlar o acesso ao servidor Web: negando o acesso a um cliente que deseja conexão com o servidor com base no seu endereço IP, ou proibindo o acesso a um cliente até que ele produza alguma forma de identificação, tipicamente um nome de usuário e a senha correspondente. Nessa segunda abordagem, uma vez conectado o usuário fica sujeito à proteção normal de diretórios através de listas de acesso.

A segurança do WWW também depende da **segurança das informações que trafegam através da Internet**. A proteção dessas informações no suporte de comunicação envolve o que poderíamos chamar de controles criptográficos. Um dos protocolos criptográficos disponíveis para uso na Internet que pode ser citado é o SSL (*Secure Socket Layer*) (RUBIN, GEER, *et al.*, 1997).

Os **clientes** (*browsers*) também podem sofrer ataques. Linguagens e ambientes de programação para a Web como Java e Javascript, com suas novas características impostas para melhorar a interatividade do ambiente WWW, também criaram problemas de segurança adicionais, os quais já foram em parte abordados no item 3.2.1.

3.3 Padrão CORBA (*Common Object Request Broker Architecture*)

As especificações CORBA/OMG definem características de um ambiente de suporte à aplicações distribuídas segundo a arquitetura OMA (*Object Management Architecture*). Essa arquitetura divide o espaço de objetos distribuídos em três partes: objetos de aplicação, construídos pelo programador de aplicação; objetos de serviço (COSS - *Common Object Services Specification*), que suportam serviços comuns, independentes de domínios de aplicação e, ainda, facilidades comuns (*Common Facilities*), que por sua vez são serviços orientados para domínios de aplicações. Os objetos se comunicam nessa arquitetura via ORB (*Object Request Broker*). O ORB, num sentido mais genérico, é um canal de comunicação para objetos distribuídos.

O padrão CORBA pode ser usado para integrar aplicações distribuídas, aumentando a reusabilidade, a portabilidade e a interoperabilidade de *softwares* orientados a objetos. Esta arquitetura torna mais fácil a programação, permitindo assim criar aplicações heterogêneas sobre as principais plataformas de *hardware* e sistemas operacionais disponíveis. O CORBA tem ainda por objetivo encorajar os desenvolvedores de aplicações abertas, para que essas sejam usadas como componentes dos sistemas de larga escala (OMG, 1999b, SIEGEL, 1998).

As especificações CORBA introduzem um conjunto de conceitos e mecanismos padronizados que permitem a implementação de objetos distribuídos. O modelo de interação adotado é o cliente/servidor. A arquitetura CORBA é constituída por um núcleo de ORB que implementa abstrações e semânticas relacionadas com a comunicação entre objetos de um sistema distribuído. O CORBA fornece os mecanismos de representação de objetos distribuídos e de suporte para comunicação. A Figura 3.1 ilustra os componentes do ORB.

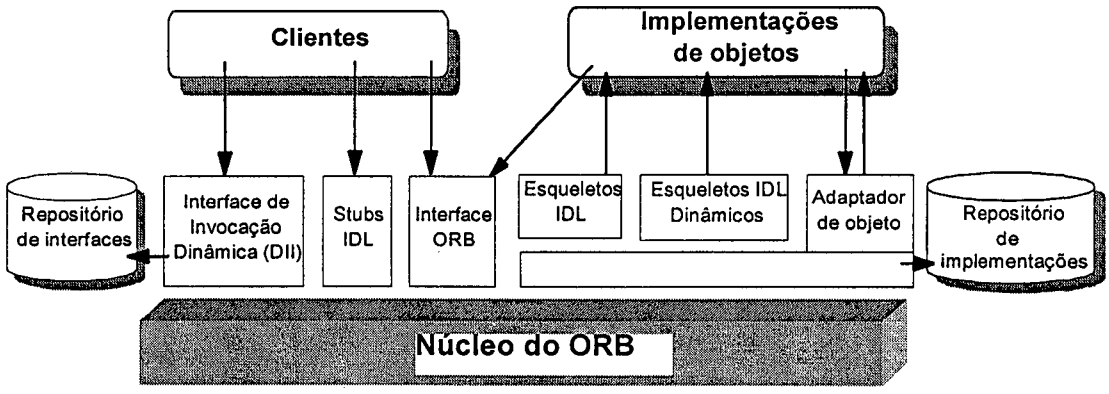


Figura 3.1 – A estrutura CORBA 2.0.

Em um ambiente CORBA, cada objeto tem sua interface padrão especificada em IDL (*Interface Definition Language*). O cliente pode requisitar serviços através de *stubs* IDL, gerados estaticamente no processo de compilação de especificações IDL de interfaces, ou construir um

pedido de serviço (invocação de método) usando um *stub* genérico de chamada (interface dinâmica DII – *Dynamic Invocation Interface*) e um repositório de interfaces. O depósito de interfaces (*Interface Repository*) permite o acesso em tempo de execução a informações relativas aos serviços implementados em objetos servidores.

Os dois tipos de interfaces (*stubs* estáticos e dinâmicos), quando usados no envio de uma requisição através de um ORB ativam as mesmas estruturas do lado do servidor. Uma requisição transferida através do ORB, determina a transferência de controle a uma implementação de objeto usando um esqueleto IDL, próprio ao adaptador do objeto que deve processar a operação desejada.

Na execução de uma requisição, a implementação de objeto pode acessar os serviços oferecidos pelo ORB através do adaptador de objeto. Os serviços oferecidos através desse adaptador, geralmente apropriados para classes específicas de objetos, podem ser: geração e interpretação de referências de objetos, invocação de métodos, ativação e desativação de objetos e implementações, mapeamento de referência de objeto em implementação, e assim por diante. As interfaces ORB e adaptadores de objetos são disponíveis para o gerenciamento no modelo.

A interface de esqueletos dinâmicos (DSI – *Dynamic Skeleton Interface*) é utilizada para interconexão entre ORBs. Quando o adaptador recebe uma invocação de um objeto para o qual não possui esqueletos, esta invocação passa, através da interface DSI, para a interface DII de um outro ORB que possua esqueletos para o objeto, e que fará a invocação.

O padrão CORBA 2.3.1 (OMG, 1999b) inclui uma especificação para interoperabilidade entre ORBs chamada ***General Inter-ORB Protocol*** (GIOP). Essa é uma descrição da forma na qual os tipos IDL são dispostos em um formato de mensagem para pedidos de chamadas (Figura 3.2).

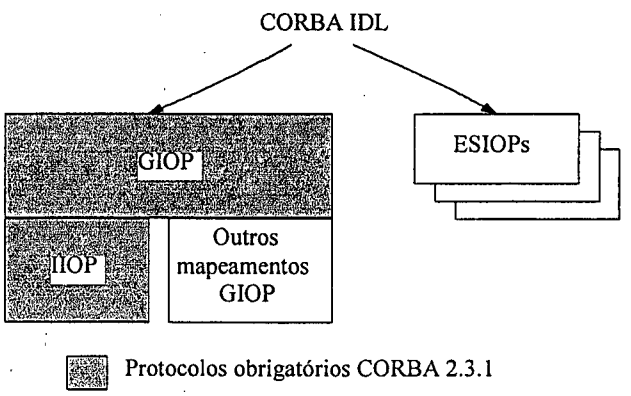


Figura 3.2 – Protocolos de Interoperabilidade do ORB.

O formato de mensagem GIOP é usado como base para a especificação do protocolo *Internet Inter-ORB Protocol* (IIOP), que usa o TCP como seu protocolo de transporte. O protocolo IIOP é obrigatório em sistemas compatíveis com o CORBA 2.3.1 e, atualmente, é o protocolo aberto Internet para comunicação entre objetos e aplicações. O CORBA 2.3.1 também fornece uma

especificação para *Interoperable Object References* (IORs – Referências de Objetos Interoperáveis), que permite a uma referência de objeto de qualquer ORB compatível ser utilizada por um cliente usando qualquer outro ORB.

3.4 O modelo CORBA de Segurança (*CORBAsec*)

O modelo CORBA de segurança (*CORBAsec*) é uma especificação aberta para segurança em sistemas de objetos distribuídos. Em maio de 1996, a OMG adotou a primeira especificação de segurança CORBA completa (CORBAsec 1.1). Depois de algumas revisões, está agora na versão 1.5 (OMG, 2000a).

O modelo de segurança descrito neste documento estabelece vários procedimentos que tratam da identificação e autenticação, da autorização e controle de acesso na invocação de um método remoto, da segurança na comunicação entre objetos (para garantir integridade e confidencialidade das mensagens), além de aspectos envolvendo esquemas de delegação de direitos, não-repudiação, auditoria e administração da segurança (OMG, 2000a).

O objetivo principal do ***CORBAsec*** é fornecer segurança no ambiente do ORB na forma de **objetos de serviços *COSS*** (***Common Object Services Specification)*** (ALIREZA, LANG, *et al.*, 2000, OMG, 2000a). Uma especificação de objeto de serviço usualmente consiste de um conjunto de interfaces e de uma descrição do comportamento do serviço, que podem ser utilizados por objetos de aplicação e outros objetos de serviço. A sintaxe usada para especificar as interfaces é a OMG IDL. A semântica que especifica o comportamento do serviço é expressa, em geral, em termos de objetos, operações e tipos de dados padronizados (OMG, 2000a).

A arquitetura de segurança CORBA foi projetada comprometendo-se a atender os seguintes requisitos gerais de segurança para aplicações distribuídas, que concretizam as propriedades fundamentais de segurança (**confidencialidade, disponibilidade, integridade, autenticidade** e **contabilização**) (LANG e SCHREINER, 2000, OMG, 2000a):

- **Transparência**: a segurança do sistema deve interferir o mínimo possível nas atividades do usuário.
- **Independência de mecanismo**: o CORBAsec deve ser independente da tecnologia de segurança subjacente.
- **Flexibilidade**: o CORBAsec deve suportar uma variedade de políticas e mecanismos de segurança.
- **Interoperabilidade**: o CORBAsec deve suportar a interoperabilidade entre implementações diferentes, entre ORBs diferentes, entre sistemas com e sem segurança, entre tecnologias de

segurança diferentes, e entre domínios de um sistema distribuído, que pode ainda suportar diferentes políticas de segurança.

- **Escalabilidade**: o sistema seguro deve se adaptar para ambos sistemas de pequena e larga escala, isto é, deve fornecer suporte ao agrupamento de sujeitos e objetos usando grupos, *roles* e domínios de políticas gerenciáveis que podem interoperar.
- **Simplicidade**: o sistema de segurança deve ser fácil de entender, usar e administrar.
- **Desempenho**: a segurança não deve impor um desempenho inaceitável.
- **Certificação**: deve ser possível avaliar e certificar a segurança do sistema seguro e dos mecanismos de segurança subjacentes de acordo com critérios de avaliação de segurança padronizados (por exemplo, ITSEC ou *Common Criteria ISO 15408*).

Alguns desses requisitos são conflitantes entre si. Um exemplo bastante discutido é o dilema existente entre a **flexibilidade** e **interoperabilidade** (GOLLMANN, 1999, LANG e SCHREINER, 2000). O problema fundamental é que a flexibilidade é atingida quando a adaptação e a extensão de serviços é permitido, enquanto a interoperabilidade requer a concordância por padrões comuns de interfaces, representações de dados e protocolos. A padronização, por sua natureza, limita a flexibilidade (GOLLMANN, 1999).

Uma arquitetura de segurança deve confinar a funcionalidade chave de segurança em um **núcleo confiável**, que garante a aplicação da parte essencial da política de segurança definida. Os ORBs seguros não necessitam suportar todas as funcionalidades de segurança definidas na especificação do CORBAsec. A especificação define dois níveis de funcionalidade em que esses ORBs podem se enquadrar de acordo com os serviços que estes dispõem[13].

O modelo CORBA de segurança, pode ser visto sob três pontos de vista diferentes: **da aplicação**, que compreende o uso das interfaces disponíveis para aplicação, **da administração**, que compreende o uso das interfaces para definir e gerenciar as políticas de segurança e domínios e, **da implementação**, que compreende o uso das interfaces disponíveis para o implementador do sistema de objetos seguro.

[13] O **nível 1** (*SecurityLevel1*) fornece segurança às invocações entre cliente e objeto destino, com delegação simples de atributos de segurança para as aplicações que não são cientes da segurança, garantindo controle de acesso aplicado pelo ORB e auditoria de eventos do sistema. Este nível possui ainda uma interface pela qual as aplicações, cientes das características de segurança, podem acessar atributos de segurança, que poderão ser aplicados nas políticas de segurança da própria aplicação (por exemplo, controle de acesso). A funcionalidade de não-repudiação é opcional.

O **nível 2** de funcionalidade (*SecurityLevel2*) suporta, além das funcionalidades de nível 1, outras interfaces de aplicação e interfaces de administração. Facilidades para a administração de políticas de segurança são introduzidas, permitindo que aplicações determinem suas próprias políticas de autorização. Opcionalmente, este nível pode ainda fornecer não-repudiação e interoperabilidade segura.

3.4.1 Modelo Estrutural do CORBAsec

O modelo CORBA de segurança relaciona objetos e componentes de quatro níveis numa estratificação de sistema (Figura 3.3): o *nível de aplicação* com os objetos de aplicação; o *nível de middleware* formado por objetos de serviço, serviços ORB e o núcleo do ORB (*nível middleware* CORBA); o *nível de tecnologia de segurança* formado pelos serviços de segurança subjacentes; e, por fim, *o nível de proteção básica* composto por uma combinação de funcionalidades de sistemas operacionais e do *hardware*. A Figura 3.3 ilustra os níveis e os componentes principais do CORBAsec.

Nesta figura, os objetos cliente e destino representam o nível das aplicações. Os serviços ORB e objetos de serviço (serviços COSS) são construídos sobre o núcleo ORB e estendem as funções básicas com qualidades ou controles adicionais, facilitando a implementação de objetos distribuídos. Um conjunto de serviços ORB e serviços COSS é usado no nível de *middleware*, na implementação da segurança no modelo CORBA. A tecnologia de segurança subjacente define algoritmos e protocolos usados na implementação de alguns objetos de serviço do *CORBAsec*. A seguir serão descritos cada um desses componentes.

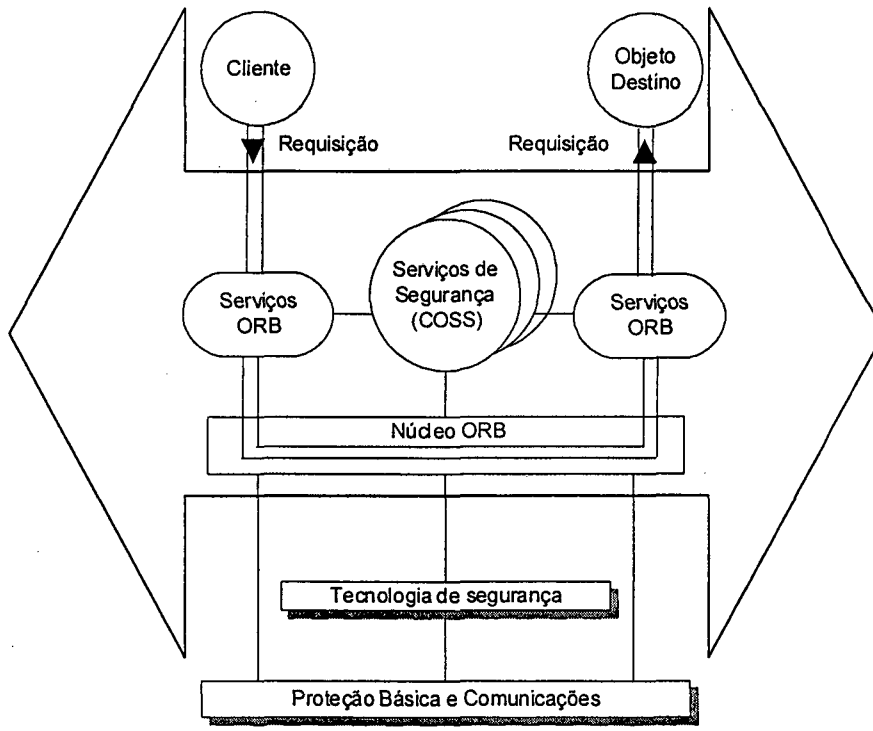


Figura 3.3 – Modelo estrutural do CORBAsec.

3.4.1.1 Nível de aplicação

Os objetos cliente e destino da Figura 3.3 representam o nível de aplicação. Muitos componentes de aplicação não são cientes da segurança e confiam no ORB para chamar, de forma transparente, os serviços de segurança requeridos durante a invocação de objetos. Entretanto, algumas aplicações garantem sua própria segurança ativando diretamente os serviços de segurança. O CORBAsec contempla mecanismos adequados às duas abordagens de implementação da segurança.

3.4.1.2 Nível de middleware CORBA

O núcleo do ORB suporta a funcionalidade mínima para habilitar um cliente a invocar uma operação de um objeto destino. Os serviços ORB e os objetos de serviço COSS de segurança do modelo da Figura 3.3 são construídos sobre o núcleo do ORB, e estendem as funções básicas de comunicação com qualidades ou controles adicionais, facilitando assim a implantação de funcionalidades como as necessárias para a segurança. Uma combinação de serviços ORB e serviços COSS é usada na implementação da segurança no modelo CORBA.

Nas especificações do modelo de segurança CORBA, os chamados *serviços ORB* são formados pelos *interceptors*. Logicamente, um *interceptor* é interposto no caminho de uma chamada entre um cliente e um destino. Cada serviço COSS relacionado com a segurança é associado a um *interceptor*, cuja finalidade é provocar o desvio transparente de uma invocação de método, ativando um objeto de serviço associado.

No modelo CORBA de segurança são definidos dois *interceptors* que atuam durante uma invocação de método (Figura 3.4): o *Access Control Interceptor* que em nível mais alto provoca um desvio para realizar o controle de acesso na chamada, e o *Secure Invocation Interceptor* que faz uma interceptação de mais baixo nível no sentido de fornecer propriedades de integridade e de confidencialidade nas transferências de mensagens correspondentes à invocação. Esses interceptadores atuam, tanto no lado cliente da invocação, como no lado do objeto servidor da aplicação.

Estes interceptadores definidos no *CORBAsec* são criados durante o processo de *binding* (ligação) entre dois objetos de aplicação que devem se comunicar. Durante a ligação a estrutura de segurança que deve atuar durante as invocações de métodos é construída no ORB. Os dois interceptadores citados estão ligados a diferentes funcionalidades em momentos distintos de uma invocação de método.

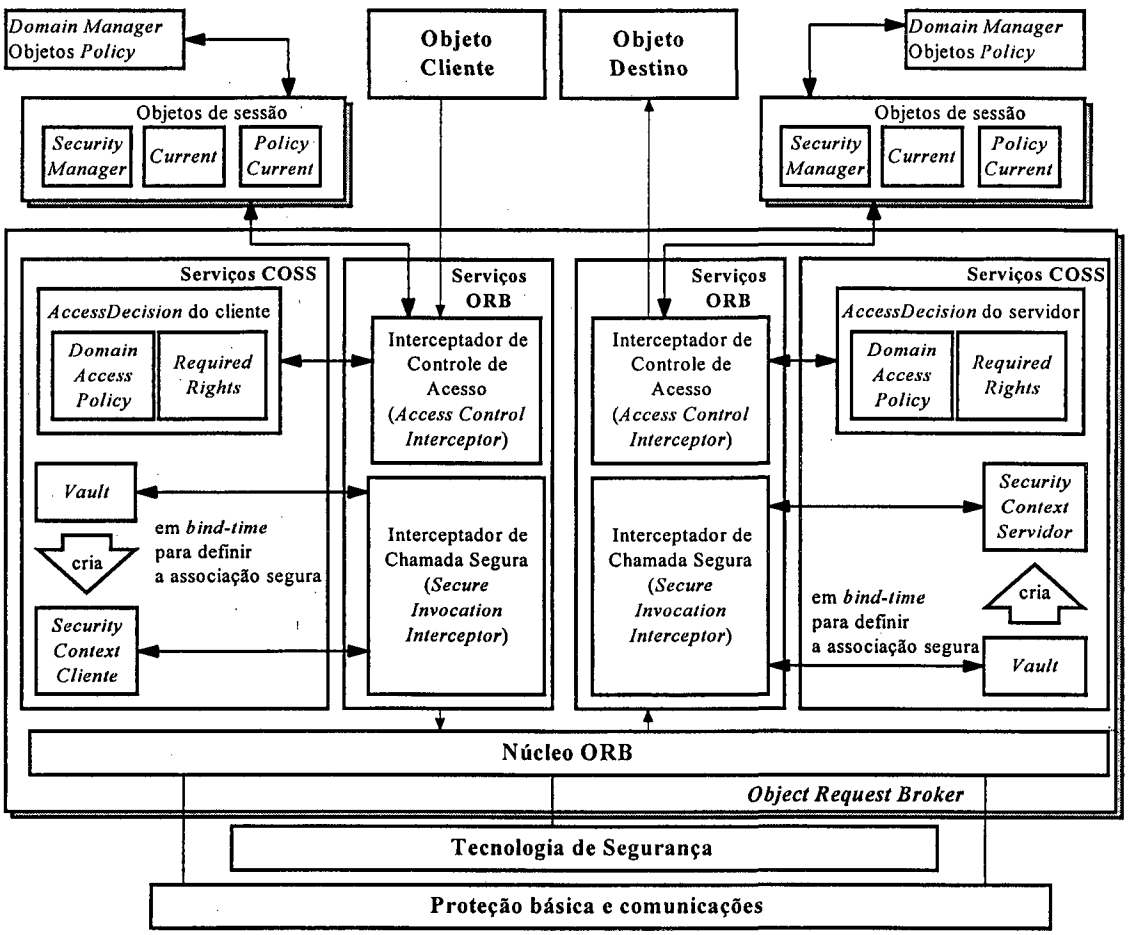


Figura 3.4 – Serviços de Segurança do Modelo CORBAsec.

Os objetos de serviço COSS que implementam os controles de segurança em uma invocação de método são: *PrincipalAuthenticator*, *Credentials*, *DomainAccessPolicy*, *RequiredRights*, *AccessDecision*, *SecurityManager*, *PolicyCurrent*, *Current*, *Vault* e *SecurityContext*. Cada um desses objetos é descrito com mais detalhes na apresentação das funcionalidades de segurança do CORBAsec. Existem, também, outros objetos de serviço relacionados com a não-repudiação e auditoria.

3.4.1.3 Tecnologia de segurança

Como apresentado na Figura 3.3, os objetos de serviço no modelo CORBA de segurança isolam aplicações e o ORB da tecnologia de segurança. Esta última consiste em uma camada subjacente que implementa várias funcionalidades de objetos de serviço do CORBA relacionados com a segurança. A tecnologia de segurança pode incluir serviços subjacentes de autenticação, de certificados e serviços de associação segura (distribuição de chaves, cifragens/decifragens). Várias tecnologias podem fornecer estes serviços e, além disso, pode-se ainda utilizar um conjunto de

interfaces genéricas, tais como a GSS-API (*Generic Security Services* API) (LINN, 1997), para isolar as implementações dos serviços de segurança CORBA do conhecimento detalhado dos mecanismos subjacentes. É importante ressaltar que o CORBAsec só fornece interfaces padronizadas para os serviços de segurança. Os mecanismos de segurança (implementações) são fornecidos por esta tecnologia de segurança subjacente. Até o momento, foram considerados na especificação do CORBAsec o uso do Kerberos, SPKM (*Simple Public-Key GSS-API Mechanism*), Sesame e SSL como tecnologias de segurança (ALIREZA, LANG, *et al.*, 2000, OMG, 2000a).

3.4.1.4 Proteção básica e comunicações

A proteção básica diz respeito ao fornecimento de meios para proteger componentes da aplicação uns dos outros, bem como proteger os componentes que suportam os serviços de segurança no ambiente operacional.

Se no ambiente operacional, que inclui o sistema operacional e o suporte para comunicação remota, existem serviços seguros de transporte de mensagens, a manipulação da criptografia (a partir da tecnologia de segurança) não é necessária no nível do ORB.

Em geral, deve-se estabelecer limites de proteção em torno de grupos de componentes que pertencem a um domínio de proteção. Componentes pertencentes a um mesmo domínio de proteção (mesmo sistema operacional e mesmo *hardware*) não possuem suas interações mediadas pelos serviços de segurança CORBA; estas interações são controladas através do uso de mecanismos de sistema operacional.

3.4.2 Autenticação no modelo de segurança CORBA

Uma entidade ativa deve primeiro estabelecer seus direitos para então tentar acessar objetos através de um ORB seguro. O **principal** é a entidade (usuário ou processo) responsável e autorizada pela política de segurança a acessar informações ou recursos mantidos pelo sistema. Este pode ser autenticado de inúmeras maneiras. A informação de autenticação mais comum para usuários humanos é uma senha, e para entidades do sistema é uma chave (*long term key*). Se um principal, ainda não autenticado, deseja usar serviços no sistema, a autenticação, como apresentada na Figura 3.5, se faz necessária.

Primeiramente, o usuário fornece a sua identidade e os dados de autenticação (por exemplo, uma senha) ao ***User Sponsor***, que representa o ponto de entrada para o usuário no sistema. É o código que chama as interfaces CORBAsec para a autenticação de usuário. Não precisa ser um objeto e não tem nenhuma interface definida. É descrito para que o processo de aquisição de credenciais possa ser melhor compreendido.

O objeto *PrincipalAuthenticator* implementa o serviço de autenticação de usuários no CORBA. A autenticação define um conjunto de trocas entre o principal e o objeto de serviço *PrincipalAuthenticator*, que tem como objetivo final a aquisição de *credenciais* para o contexto do principal. A aquisição de credenciais é feita de forma transparente para as aplicações que não tem conhecimento sobre a segurança envolvida em um ambiente CORBA.

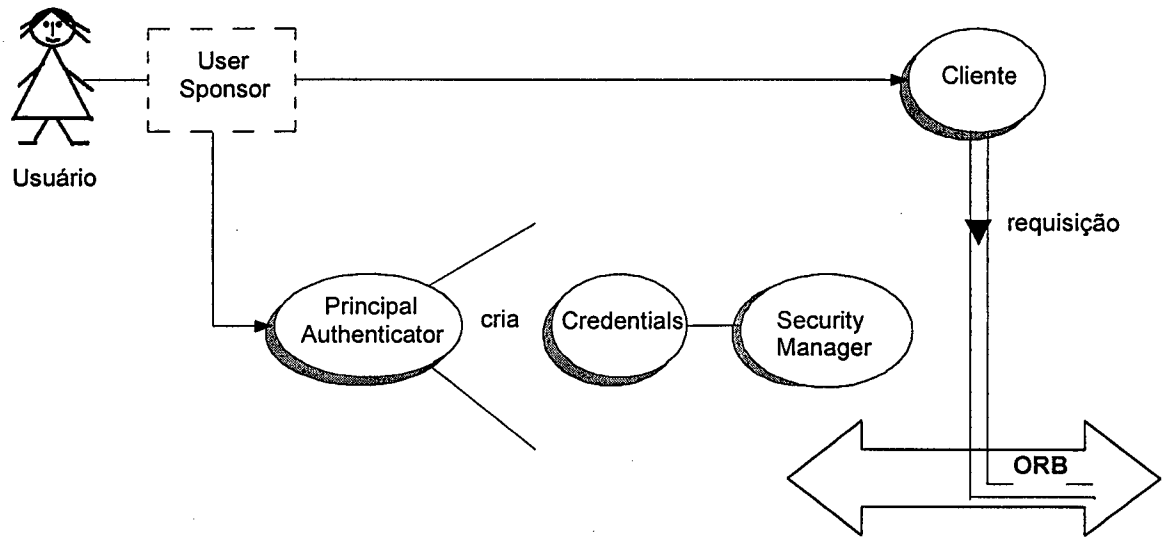


Figura 3.5 – Modelo para Autenticação.

Uma credencial (objeto *Credentials*), além de privilégios de identidade que designam o principal no sistema, contém os atributos de privilégios necessários para que o principal possa acessar determinados objetos e serviços no sistema durante uma sessão. A Figura 3.6 mostra o objeto credencial que armazena os atributos de segurança de um principal.

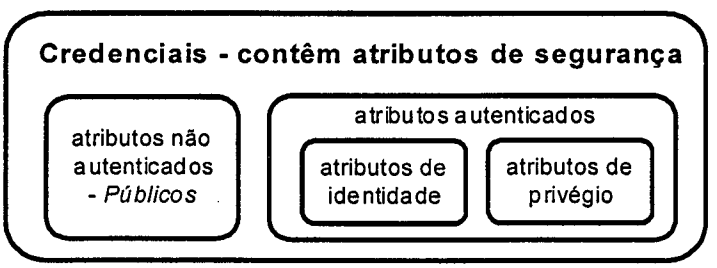


Figura 3.6 – Credenciais.

Alguns atributos de privilégio enfatizam o compartilhamento entre usuários. É o caso de privilégios *groups*, *roles*, *clearances* e *capabilities*. Existe ainda o atributo de privilégio que inclui a identidade de acesso do principal (*AccessId*).

As credenciais podem ser:

- ***OwnCredentials***: representam as credenciais da aplicação criadas por um processo de autenticação;

- ***ReceivedCredentials***: o objeto destino deve obter o objeto ***ReceivedCredentials*** para identificar o pedido do principal, e obter os privilégios do mesmo.

Um objeto ***ReceivedCredentials*** representa uma informação do principal remoto, incluindo as mesmas informações do objeto ***OwnCredentials***, como atributos de privilégio e identidade do principal remoto. O objeto ***ReceivedCredentials*** pode também ser usado nas invocações com delegação de privilégios.

Depois de adquirir credenciais, o principal e os objetos que atuam em seu nome como clientes podem iniciar suas chamadas a objetos servidores no sistema.

O objeto ***SecurityManager*** da Figura 3.5 representa o contexto de execução atual que mantém as informações do estado de segurança dos objetos cliente e destino, como por exemplo a credencial criada em favor do principal.

Os objetos *PrincipalAuthenticator* e *Credentials* se encontram no pacote *SecurityLevel2* na especificação do CORBAsec.

3.4.2.1 Objetos de Sessão

Objetos cruciais para o entendimento da dinâmica do funcionamento do CORBAsec são os chamados *objetos de sessão* (Figura 3.4). Os objetos de sessão ***SecurityManager***, ***PolicyCurrent*** e ***Current*** armazenam informações sobre o contexto corrente de segurança relativo ao processo (*capsule specific*) ou ao fluxo de execução da invocação (*thread specific*), respectivamente (ADIRON, 1999, OMG, 2000a). O objeto ***SecurityManager*** mantém informações de estado associadas com a instância do ORB (*thread specific*), ou com o processo no qual o ORB reside (*capsule specific*). Por exemplo, referências aos objetos ***RequiredRights*** e ***AccessDecision***, que são utilizadas durante uma invocação, e ao conjunto de objetos ***Credentials*** do principal, são armazenadas no objeto *SecurityManager*. O objeto ***Current*** armazena, no lado do servidor, as credenciais que são enviadas pelo cliente durante o estabelecimento de uma associação segura.

As interfaces ***SecurityLevel1::Current*** e ***SecurityLevel2::SecurityManager*** permitem o acesso às informações de segurança associadas com o contexto de execução corrente e às credenciais da sessão. Os serviços de segurança usam os objetos ***Current*** e ***SecurityManager*** para obter os atributos de privilégios de um principal, obter informações relativas a uma associação segura e obter as referências dos objetos ***RequiredRights*** e ***AccessDecision***. Além disso, o objeto destino também pode usar os objetos ***Current*** e ***SecurityManager*** para fornecer decisões de acesso adicionais dependentes do contexto.

3.4.3 Domínios de Políticas de Segurança no CORBAsec

O problema do grande crescimento do número de **objetos** (*scale*) e a implementação de políticas de segurança aos acessos a objetos, em sistemas de objetos distribuídos, é resolvido no CORBAsec pelo **agrupamento** destes objetos em **domínios de políticas.**

Um ***domínio*** (*domain*) é um **conjunto de objetos** que compartilham uma característica comum ou que respeitam um conjunto comum de regras. O CORBAsec define vários tipos de domínios: ***security policy domains***, ***security environment domains*** e ***security technology domains*** (OMG, 2000a).

De acordo com o modelo CORBA de segurança, quando um objeto é criado este automaticamente torna-se membro de um ou mais domínios (domínios de políticas), ficando sujeito às políticas de segurança do domínio. Entretanto, a especificação atual (OMG, 2000a) não possui procedimentos para gerenciar os membros dos domínios (incluir, remover, etc) e nem mesmo possui procedimentos para gerenciar os objetos de políticas de segurança. Os **domínios de políticas de segurança** (*security policy domains*) são constituídos por coleções de referências de objetos que têm acessos a seus métodos regrados por um conjunto de políticas de segurança comuns (objetos ***Policy***). Estes domínios de políticas são gerenciados por um objeto *gerente de domínio* (objeto ***DomainManager***) (OMG, 2000f) (Figura 3.7). Este gerente tem como funções (OMG, 1999a): prover mecanismos para a criação e o acesso aos objetos de política no domínio; e também, manter as estruturas do domínio atualizadas.

Podem existir várias políticas de segurança associadas com um domínio, com um objeto de política para cada uma delas. Existe pelo menos uma política de cada tipo associada com cada domínio de política. Esses objetos de política são compartilhados entre os objeto do domínio, ao invés de estarem associados com objetos individuais.

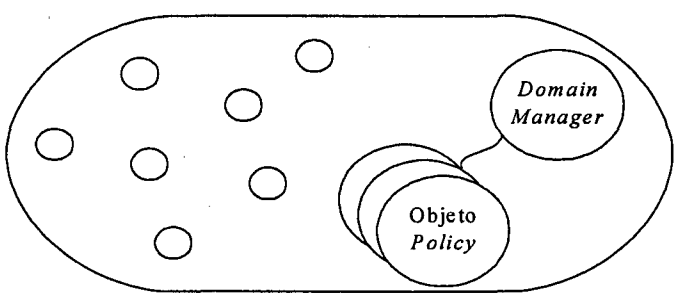


Figura 3.7 – Objetos do Domínio.

Domínios de políticas de segurança possibilitam a administração de muitos objetos com um número menor de políticas, o que auxilia a resolver o problema de gerenciamento de sistemas de

objetos distribuídos em larga escala. Além disso, a autoridade dos administradores pode ser dividida pela delimitação das atividades administrativas a um domínio ("*separation of duties*").

Entretanto, interfaces para adicionar novas políticas (novos objetos de política) aos domínios ou para modificar os membros de domínios (*membership*) ainda não estão padronizadas. A idéia inicial para preencher estas lacunas está sendo proposta no serviço de gerência de membros de domínios de segurança (*Security Domain Membership Management Service*) (OMG, 2000f), onde é proposta a extensão da interface *DomainManager* com funções administrativas que sirvam para definir e remover objetos de políticas em um domínio.

Security environment domains especificam um conjunto de objetos que pode ter os requisitos da política de segurança garantidos de forma local no seu ambiente de execução. Exemplo: cifragem pode não ser necessária quando as mensagens são transferidas entre objetos na mesma máquina. *Environment domains* não são visíveis para as aplicações ou para os serviços CORBA de segurança.

Security technology domains são domínios que usam os mesmos mecanismos de segurança (mesma tecnologia de segurança). Por exemplo, os mesmos métodos podem estar disponíveis para autenticação de *principais*; dados em trânsito podem ser protegidos da mesma forma, usando a mesma tecnologia de distribuição de chaves e de algoritmos de cifragem/decifragem. O objetivo desse tipo de domínios é identificar quais objetos usam os mesmos serviços de segurança subjacentes. Isso torna possível definir e manter os serviços subjacentes do domínio e administrar entidades da forma requerida pela tecnologia de segurança.

3.4.3.1 Associação entre os Objetos Policy e DomainManager

Um ORB ou um serviço CORBA podem permitir a definição de algumas escolhas, feitas por objetos de aplicação ou de serviço. Estas escolhas podem ser feitas, de forma estruturada, usando interfaces derivadas da *interface Policy*.

Normalmente, os objetos de política (objetos *Policy*) são associados com e usados por outros objetos do modelo CORBA de segurança. O uso dos objetos *Policy* no contexto do CORBAsec envolve sua associação com o contexto de execução bem como com os objetos *DomainManagers*.

Associação dos objetos de política com o contexto de execução

Informações de estado associadas ao contexto de execução corrente residem no objeto *PolicyCurrent*. Importantes no estabelecimento de uma associação segura entre objetos clientes e servidores, os *objetos de Políticas de Invocação* definem características como a qualidade de

proteção das mensagens, as credenciais a serem usadas na invocação, etc. Os objetos de *Políticas da Invocação* que definem esta qualidade de proteção no CORBAsec são os seguintes:

- ***QOPPolicy***: este objeto é o portador da política de qualidade de proteção que deve ser usada para fazer uma invocação.
- ***InvocationCredentialPolicy***: este objeto é portador da política que define as credenciais que poderão ser usadas na invocação desejada.
- ***SecurityMechanismPolicy***: este objeto é portador da política que define os mecanismos que devem ser usados na invocação desejada.
- ***EstablishTrustPolicy***: este objeto é portador da política que contém a orientação para o estabelecimento da confiança do cliente pelo servidor e do servidor pelo cliente.
- ***DelegationDirectivePolicy***: este objeto é portador da política que informa se a delegação pode ser usada durante a invocação.

Esses objetos de política são criados para o processo (*process/capsule*)/instância do ORB quando a aplicação é instanciada. São armazenados no objeto *PolicyCurrent* que fornece operações para ajustar essas políticas a fim de que sejam utilizadas 'por *thread* de execução' (*per-execution thread basis*). O cliente e o servidor possuem um objeto *PolicyCurrent* que armazena as políticas usadas na *thread* de execução corrente em cada um dos lados.

Associação dos objetos de política com os objetos *DomainManagers*

Na criação de uma referência de um objeto, o ORB implicitamente associa a referência do objeto com um ou mais domínios de política.

Um domínio de política inclui um único objeto, um por domínio de política, chamado de gerente do domínio (*domain manager*). O gerente do domínio registra os membros do domínio e fornece meios para a adição e remoção de membros. O próprio gerente do domínio é um membro do domínio, possivelmente do domínio que ele gerencia.

Existem várias políticas associadas com um domínio, com um objeto *Policy* para cada uma. Os objetos de política determinam controles sobre os objetos do domínio.

Gerentes de domínios (*Domain Managers*) e objetos de política (*Policy*) possuem dois tipos de interfaces:

- *Interfaces operacionais usadas para implantar as políticas*: Essas interfaces são usadas pelo ORB durante uma invocação de um objeto. Alguns objetos de política podem também ser usados pelas aplicações que garantem suas próprias políticas;
- *Interfaces administrativas usadas para definir as políticas* : Interfaces usadas para especificar, por exemplo, os eventos a serem verificados (*audited*) ou quem tem acesso aos objetos de um

tipo específico nesse domínio. O administrador vê e navega pelas estruturas de domínios, assim ele é consciente do escopo do que ele está administrando.

3.4.4 Definição das Políticas de Segurança no CORBAsec

O modelo administrativo descrito na especificação do CORBAsec explica como definir e administrar as políticas de segurança. As atividades administrativas descritas pela especificação são as seguintes (OMG, 2000a):

- **criar** objetos em um ambiente seguro sujeito às políticas de segurança;
- **encontrar** os **gerentes de domínios** de um objeto específico;
- **encontrar** os **objetos de política** pelos quais os gerentes de domínios são responsáveis;
- **definir** os detalhes de **política** para esses objetos de política;
- **especificar as políticas de autorização** que definem os direitos de acesso no domínio.

Conforme o módulo *Security.IDL* do CORBAsec (OMG, 2000a), as seguintes constantes são usadas para identificar os diferentes tipos de políticas de segurança:

```
// General administrative policies
const CORBA::PolicyType SecClientInvocationAccess = 1;
const CORBA::PolicyType SecTargetInvocationAccess = 2;
const CORBA::PolicyType SecApplicationAccess = 3;
const CORBA::PolicyType SecClientInvocationAudit = 4;
const CORBA::PolicyType SecTargetInvocationAudit = 5;
const CORBA::PolicyType SecApplicationAudit = 6;
const CORBA::PolicyType SecDelegation = 7;
const CORBA::PolicyType SecClientSecureInvocation = 8;
const CORBA::PolicyType SecTargetSecureInvocation = 9;
const CORBA::PolicyType SecNonRepudiation = 10;

// Policies used to control attributes of a binding to a target
const CORBA::PolicyType SecMechanismsPolicy                = 12;
const CORBA::PolicyType SecInvocationCredentialsPolicy     = 13;
const CORBA::PolicyType SecQOPPolicy                       = 15;
const CORBA::PolicyType SecDelegationDirectivePolicy       = 38;
const CORBA::PolicyType SecEstablishTrustPolicy            = 39;
```

Pela especificação, para gerenciar as políticas de segurança, uma aplicação administrativa realiza as operações representadas na Figura 3.8.

Quando uma nova referência de objeto é criada ela é automaticamente associada com um domínio de política de segurança. Uma vez que um gerente de domínio tenha sido criado, novos objetos de política podem ser associados com esse gerente, usando facilidades de gerência de domínios apropriadas.

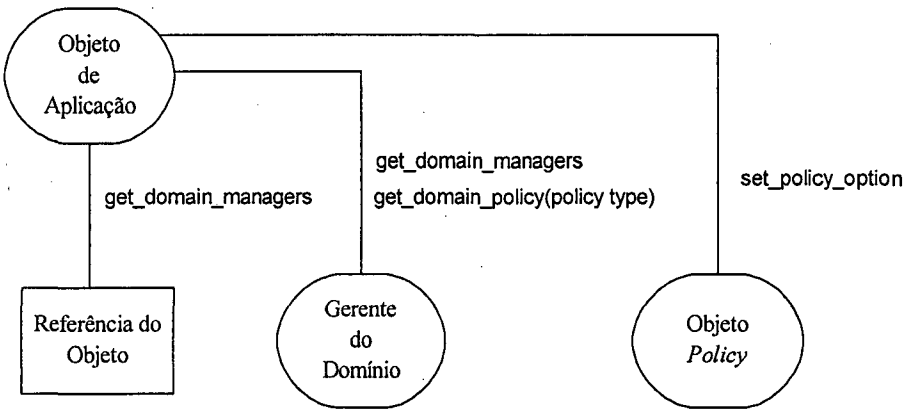


Figura 3.8 – Gerenciando Políticas de Autorização.

Uma aplicação pode invocar a operação *get_domain_managers* a partir da referência do objeto para obter uma lista de gerentes de domínios para aquele objeto, isto é, objetos gerentes de domínio associados com a referência do objeto destino.

Tendo encontrado um gerente de domínio, a aplicação administrativa pode então encontrar as políticas de segurança associadas com aquele domínio chamando a operação *get_domain_policy* do gerente de domínio, especificando o tipo de política que se quer encontrar (por exemplo, *client invocation access policy*, *target invocation access policy*). A operação *get_domain_policy* retorna o objeto *Policy* necessário para administrar a política associada com esse domínio. Cada objeto *Policy* suporta as operações requeridas para administrar o tipo de política especificado.

Ao ter encontrado o objeto *Policy* requerido, a aplicação usa sua interface para definir a política. Por exemplo, o objeto *DomainAccessPolicy* (política de autorização do domínio) possui operações específicas para lidar com os direitos de acesso.

3.4.5 Controle de Acesso no modelo de segurança CORBA

Depois do processo de autenticação dos clientes e da criação do objeto *Credential*, tendo-se definido os objetos de política de autorização do ambiente, acontece o processo de autorização. O modelo CORBA de segurança permite definir políticas de autorização usando mecanismos de ACLs (*Access Control Lists*), listas de *capabilities* e controle de acesso baseado em *roles* (RBAC). A especificação provê suporte direto a políticas de autorização discricionárias através dos objetos *DomainAccessPolicy* e *RequiredRights* e usa listas de controle de acesso para armazenar os direitos de acesso. A especificação não provê suporte direto ao uso de *capabilities*, e o uso de *Role Based Access Control* é feito de forma simplificada através dos atributos de privilégio do tipo *role*.

O modelo de controle de acesso do CORBAsec é representado na Figura 3.9.

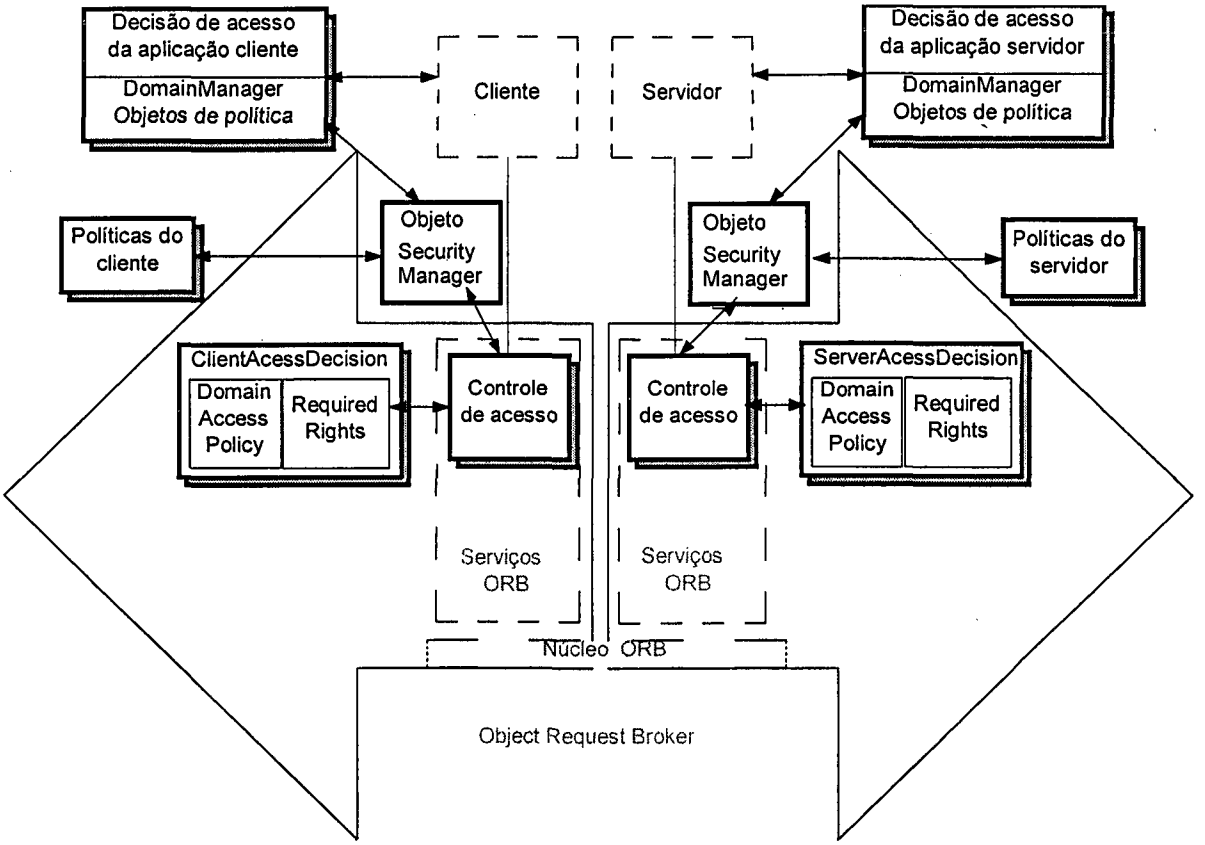


Figura 3.9 – Objetos que atuam no controle de acesso.

Os controles de autorização enfatizados no CORBAsec permitem que políticas de autorização definidas sejam verificadas em dois níveis: **no nível de *middleware*** usando objetos de serviço durante uma invocação de método (*object invocation access policy*) e, portanto, de forma transparente para a aplicação (*security unaware*); e **no nível da aplicação** (*application object access policy*), que fica consciente dos serviços de segurança do ambiente (*security aware*). Neste trabalho nós nos restrigimos aos controles de autorização transparentes (*security unaware applications*).

O modelo de autorização é baseado no uso de funções de decisão de acesso (objetos *Access Decision*), que decidem se uma operação ou função pode ser executada, aplicando-se as regras do controle de acesso (Figura 3.9).

Políticas de acesso específicas da aplicação (*application object access policy*) necessitam definir suas próprias funções de decisão de acesso para garantir o cumprimento das suas próprias regras de controle de acesso. A especificação não possui interfaces administrativas próprias para esses tipos de políticas, visto que aplicações diferentes possuem necessidades diferentes.

Políticas de acesso aplicadas na invocação dos objetos (*object invocation access policy)* usam para implementar suas funções de decisão de acesso os atributos de privilégio do objeto *initiator* (o *principal* que requisita a invocação), atributos de controle do objeto servidor e outras informações relevantes sobre a ação a ser realizada, como a operação e os dados ou sobre o contexto.

No *CORBAsec*, as políticas de segurança são descritas na forma de *atributos de segurança* dos recursos do sistema (*atributos de controle*) e dos principais (*atributos de privilégio*). O objeto ***DomainAccessPolicy*** representa a interface de acesso à política de autorização discricionária, representando para um conjunto de principais um conjunto de direitos na execução de operações em todos os objetos de um domínio. A representação de uma política de acesso particular é definida na Tabela 3.1. Para simplificar a administração e lidar com o problema do grande crescimento do número de sujeitos (*scalability*) em sistemas de objetos distribuídos, o *DomainAccessPolicy* agrega principais usando seus atributos de privilégio. Alguns tipos de agrupamento são *group* (grupo) e *role* (papéis desempenhados pelos principais no sistema). Existe a possibilidade de permitir diferentes direitos a um sujeito que pode iniciar uma invocação (*initiator*) ou que está participando de uma cadeia de delegação de tarefas (*delegate*). Dessa forma, o sujeito no sentido tradicional de uma matriz de acesso é a combinação de um atributo de privilégio e de um estado de delegação. Para lidar com diversidade de direitos e a correspondentes necessidades de expressar a autorização, os direitos são classificados em conjuntos de *tipos de direitos* chamados *famílias de direitos*. O *CORBAsec* fornece apenas a família *Corba* que contém quatro tipos de direitos: *g* (*get*), *s* (*set*), *m* (*manage*) e *u* (*use*), embora permita a livre definição de outras famílias de direitos.

Atributo de Privilégio	**Estado de Delegação**	**Direitos Fornecidos (*Granted Rights*)**
role:gerente_banco	Initiator	corba: gs--
role:gerente_banco	Delegate	corba: g---
role:caixa	Initiator	corba: g--u

Tabela 3.1 – Objeto *DomainAccessPolicy*.

Uma política de acesso fornece então direitos aos atributos de privilégio conforme o seu estado de delegação. Em contraste, um objeto ***RequiredRights*** define que para a invocação de cada operação na interface de um objeto seguro, alguns direitos são necessários ou requeridos (*atributos de controle*). Exemplos de atributos de controle podem ser: listas de controle de acesso que identificam usuários permitidos pelo nome ou por atributos de privilégio; informações usadas em esquemas baseados em rótulos (*label-based schemes*), como a classificação de um objeto, que identifica a habilitação (*clearance*) dos *principais* permitidos a executar operações do objeto.

Para lidar com o problema do grande crescimento do número de operações em sistemas de objetos distribuídos, o *RequiredRights* agrupa operações pela sensibilidade das mesmas usando as famílias de direitos (*rights*).

Um exemplo de objeto *RequiredRights* é mostrado na Tabela 3.2. Essa tabela define os direitos requeridos para se ter acesso a cada operação específica de um objeto. Também existe o combinador de direitos que possibilita especificar se um principal necessita todos (*All*) os direitos requeridos para executar uma operação, ou se é suficiente a posse de qualquer um (*Any*) dos direitos requeridos.

Os direitos requeridos são atribuídos para *interfaces* e não para *instâncias*, ou seja, todas as instâncias de uma interface terão sempre o mesmo conjunto de direitos requeridos.

Direitos Requeridos (*Required Rights*)	**Combinador de Direitos**	**Operação**	**Interface**
Corba:g---	All	Ver_Saldo	Renda_Fixa
Corba:gs--	Any	Depositar	Renda_Fixa
Corba:g--u	All	Depositar	Conta_corrente

Tabela 3.2 – Objeto *RequiredRights* para as interfaces Renda_Fixa e Conta_corrente.

Toda a decisão de acesso na invocação dos objetos é feita através de uma interface de um objeto de serviço *AccessDecision* que determina se uma operação a ser executada por um objeto destino específico é permitida ou não (Figura 3.10). As decisões de acesso dependem dos atributos de privilégio e de controle fornecidos, respectivamente, pelos objetos *DomainAccessPolicy* e *RequiredRights*. A regra geral na decisão de acesso pode ser enunciada como segue: um cliente recebe autoridade para executar a operação *m* em um objeto servidor *o*, se a credencial associada com a invocação do método contém um atributo de privilégio que forneça um direito em um domínio de segurança que contém o objeto *o*, e se o mesmo direito está presente na tabela "*RequiredRights*" da operação *m*. Por exemplo, a política definida na Tabela 3.1 fornece a um principal *caixa*, no estado de delegação *Initiator*, todos (*All*) os direitos requeridos - *g* e *u* - para executar a operação *Depositar* da interface *Conta_corrente*.

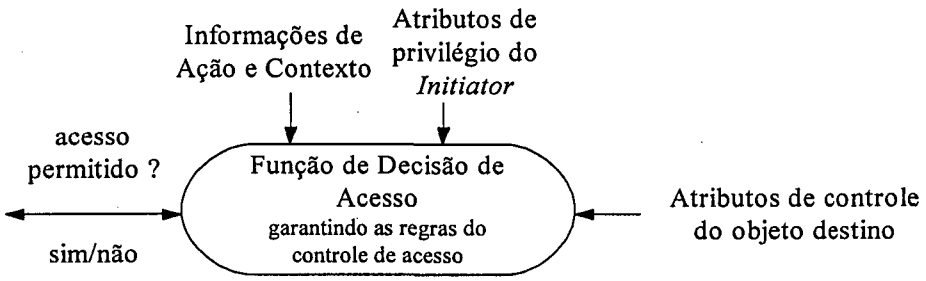


Figura 3.10 – Modelo de Autorização.

No modelo CORBAsec, as decisões de controle de acesso podem ser realizadas tanto no lado do cliente como no lado do servidor. O controle de acesso no lado do objeto destino é definido nas especificações CORBAsec como *normal*. Mas, também é salientado que, quando as verificações de controle de acesso são feitas no lado do cliente, as negações de acesso ao objeto servidor determinam uma redução no tráfego de rede. Nota-se também que em alguns ORBs, considerações de integridade do sistema podem tornar indesejável a confiança no controle de acesso feito unicamente no lado do cliente (OMG, 2000a).

Informações de estado do contexto de execução corrente, associadas aos objetos de política, residem no objeto *PolicyCurrent* (objeto de sessão, Figura 3.4). Importantes no estabelecimento de uma associação segura entre objetos clientes e servidores, os *objetos de Políticas de Invocação* definem características como a qualidade de proteção das mensagens e as credenciais a serem usadas na invocação. Objetos de *política da invocação* que determinam os tipos de associações possíveis e o objeto *DomainAccessPolicy* que atua durante uma invocação são acessados a partir do *PolicyCurrent*.

3.4.5.1 Dinâmica do Interceptador de Controle de Acesso

O **interceptador de controle de acesso** (*Access Control Interceptor*), que em nível mais alto provoca um desvio para realizar o controle de acesso na invocação, atua em dois momentos distintos: em ***bind time*** (em tempo de ligação dos objetos cliente e destino) e em ***access decision time*** (em tempo de decisão de acesso) (Figura 3.4 e Figura 3.9).

Em *bind time*, o interceptador de controle de acesso é responsável por criar o objeto *AccessDecision* registrando sua referência no objeto *SecurityManager*. O objeto *AccessDecision*, por sua vez, em *bind time* é responsável por disponibilizar os objetos de política (*DomainAccessPolicy*) do domínio. Isso ocorre como conseqüência da execução da operação *get_domain_policy* do gerente de domínio, o que resulta na inserção da referência do *DomainAccessPolicy* no objeto *PolicyCurrent*. O *AccessDecision* também localiza ainda o objeto *RequiredRights* do ambiente inserindo a referência ao mesmo no objeto *SecurityManager*.

Em *access decision time*, o interceptador de controle de acesso invoca a operação *access_allowed* do objeto *AccessDecision*. A operação *access_allowed* é responsável por autorizar ou não a invocação, obtendo os direitos fornecidos ao principal pela política de autorização *DomainAccessPolicy* (*GrantedRights*) e comparando os mesmos com os direitos requeridos (obtidos do objeto *RequiredRighs*) para executar a operação na invocação.

3.4.6 Estabelecimento da associação segura

Antes que um cliente possa usar uma referência de objeto para invocar uma operação do servidor, uma associação segura precisa ser estabelecida, associando o cliente ao objeto destino. Esta associação segura é estabelecida durante o *binding* entre objetos cliente e servidor.

A especificação (OMG, 2000a) determina que o serviço ORB, designado **interceptador de chamada segura** (*Secure Invocation Interceptor*), é responsável pelo estabelecimento do contexto de segurança requerido para proteger mensagens entre cliente e servidor, definindo o que é chamada uma associação segura. Para aplicações que não são cientes da segurança envolvida, o estabelecimento desta associação segura ocorre de forma completamente transparente.

A Figura 3.11 apresenta os serviços de segurança CORBA que participam de uma invocação segura. O interceptador de chamada segura interage com o objeto ***Vault*** que cria o objeto ***SecurityContext*** na interceptação de mais baixo nível, concretizando uma associação segura.

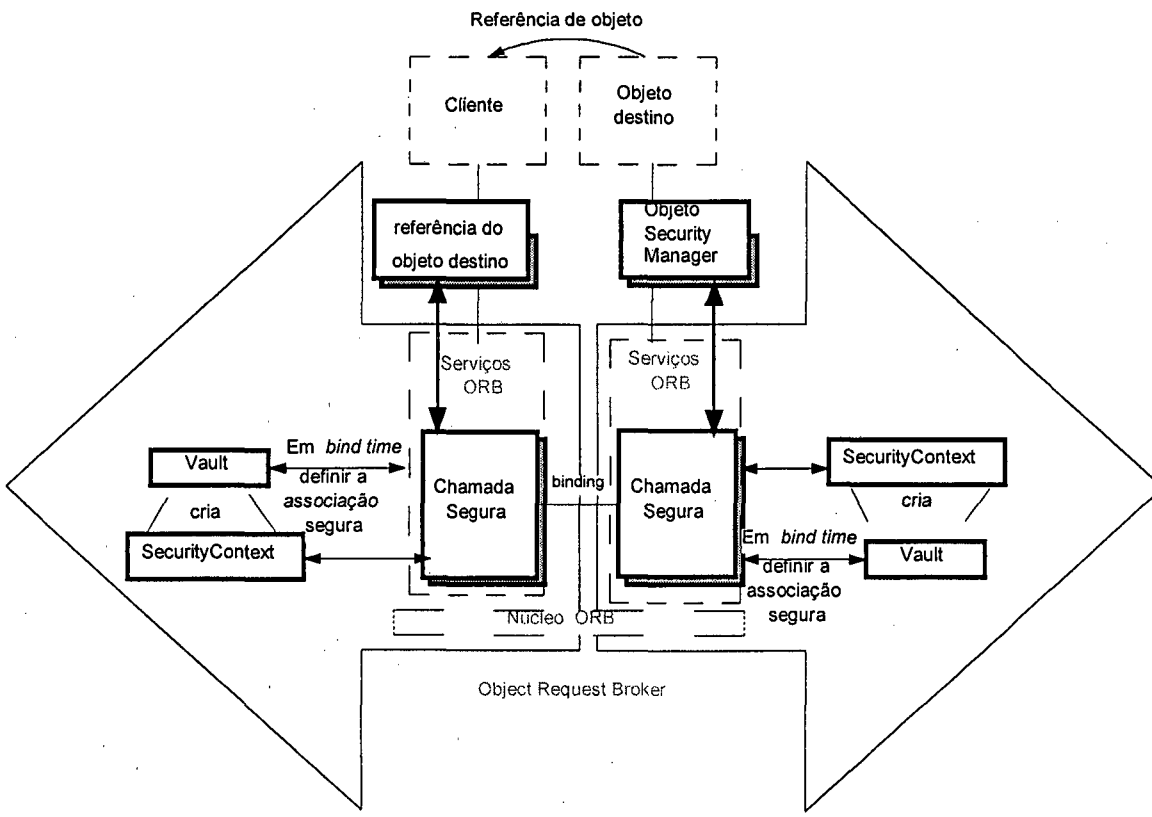


Figura 3.11 – Estabelecimento da associação segura.

O objeto ***Vault*** faz parte de um módulo substituível definido no CORBAsec, (***SecurityReplaceable***) e é responsável pela negociação da associação segura. A implementação deste objeto está diretamente relacionada à tecnologia de segurança subjacente, já que estabelecer o contexto de segurança é função deste objeto.

Todo o contexto de segurança (do cliente e do servidor), que será utilizado durante a associação segura, encontra-se nos objetos ***ClientSecurityContext*** e ***ServerSecurityContext***.

3.4.6.1 Dinâmica do Interceptador de Chamada Segura

O interceptador de chamada segura (*Secure Invocation Interceptor*), que faz uma interceptação de mais baixo nível no sentido de fornecer propriedades de integridade e de confidencialidade nas transferências de mensagens correspondentes à invocação, atua em dois momentos distintos: no instante ***bind time*** (em tempo de ligação), quando a associação segura é estabelecida, e no instante ***message protection time*** (em tempo de proteção de mensagens), quando a mensagem é protegida com mecanismos de confidencialidade e/ou integridade (Figura 3.4 e Figura 3.11).

Bind Time

De forma detalhada, toda a dinâmica do *binding* do interceptador de chamada segura pode ser dividida em quatro grandes passos (Figura 3.12), que são descritos na seqüência (WANGHAM, 2000).

Passo 1)	Define e/ou obtém as configurações necessárias para a invocação no lado do cliente, a partir dos *objetos de políticas de invocação* (***QOPPolicy***, ***InvocationCredentialPolicy***, ***SecurityMechanismPolicy***, ***EstablishTrustPolicy***, ***DelegationDirectivePolicy***).

A cada primeira invocação sobre um objeto, o ORB estabelece uma associação segura entre o cliente e o objeto alvo, usando para isto a referência do objeto destino (*IOR*). As propriedades da associação segura dependem das políticas que são identificadas na referência de objeto, usando a operação ***set_policy_overrides*** sobre a referência[14]. A associação também depende das políticas que são definidas como pertencentes ao fluxo de execução da invocação (políticas da *thread*), adicionadas usando a operação ***set_overrides*** sobre o objeto ***PolicyCurrent***[15].

Passo 2)	São analisadas as políticas para selecionar um mecanismo, estabelecer a confiança entre as partes, garantir delegação e selecionar a credencial que deve ser usada na invocação (deve ser compatível com os componentes de segurança da referência do objeto destino).

[14] Esta operação só é executada por aplicações cientes da segurança, e que desejam definir as suas políticas de invocação. Logo, estas políticas não serão consideradas.

[15] Este objeto está sendo incluído na especificação CORBAsec como *SecurityLevel2::PolicyCurrent*. Entretanto, como até este momento os mecanismos para sobrescrever políticas não foram ainda padronizados para o ***PolicyCurrent***, a exemplo do ORBAsec SL2 (ADIRON, 2000), as operações ***get_overrides*** e ***set_overrides*** são encontradas podem ser encontradas na interface que estende o *Current*.

As políticas no lado do cliente, que estão definidas para uma referência de objeto particular, empregam uma semântica particular na determinação das características de invocação feitas com a referência de objetos. O ORB executa um procedimento de decisão para determinar as características de segurança compatíveis com os mecanismos de segurança que o objeto destino suporta, anexadas na referência do objeto, e com as políticas de segurança no lado do cliente.

O procedimento de decisão aplicado pelo serviço de segurança ORB (interceptador de chamada segura) determina o mecanismo de segurança, um único objeto ***Credentials*** compatível e um componente de segurança da *IOR* do objeto destino para ser usado nas invocações feitas sobre a referência de objeto.

Passo 3)	Estabelece o contexto de segurança (objetos ***SecurityContext***) que será usado para proteção da requisição (*request*) e das respostas (*responses*).

Utilizando as características de segurança resultantes do procedimento de decisão, o interceptador chamada segura do cliente chama o método ***init_security_context*** do objeto ***Vault*** para iniciar o estabelecimento do contexto de segurança.

O objeto ***Vault*** retorna ao interceptador um *token* de segurança para ser enviado ao objeto destino e indica se haverá continuação de trocas de informações. Este também retorna uma referência para o objeto ***ClientSecurityContext***, recentemente criado para esta associação cliente/servidor.

O interceptador deve construir uma mensagem de estabelecimento de associação, incluindo o *token* de segurança (retornado pelo ***Vault***) que deve ser transferido para permitir o estabelecimento do contexto de segurança do lado do servidor. A mensagem de estabelecimento da associação pode ser construída de duas maneiras:

1) Quando apenas o *token* de segurança inicial é necessário para estabelecer a associação. Após recebida no servidor, a mensagem de estabelecimento da associação é interceptada pelo interceptor de chamada segura do servidor, que no *bind time* chama ***Vault::accept_security_context***, para criar o objeto ***ServerSecurityContext***.
2) Quando diversas trocas são requeridas para estabelecer a associação segura, a mensagem de estabelecimento de associação é enviada separadamente. O interceptador envia a mensagem para o objeto destino. Esta mensagem é interceptada no servidor pelo seu interceptador de chamada segura, que invoca, em tempo de ligação, o método ***accept_security_context*** do objeto ***Vault***, para criar o objeto ***ServerSecurityContext***. Entretanto, neste caso o interceptador do servidor deve gerar um outro *token* de segurança e emiti-lo de volta para o interceptor do cliente. O interceptador cliente chama o objeto ***ClientSecurityContext***, através da operação ***continue_security_context***.

Quando o interceptador de chamada segura recebe uma mensagem de estabelecimento de associação, este usa o ***Vault*** (se ainda não há contexto de segurança) ou o apropriado objeto contexto de segurança para processar o *token* de segurança. Quando a associação é estabelecida, a função de proteção da mensagem é usada para recuperar o *request* e proteger o *reply*.

Como já mencionado, o objeto ***Vault*** e os objetos de contexto criados por este estão diretamente relacionados com a tecnologia de segurança que será usada para fornecer os controles criptográficos. A especificação do CORBAsec sugere o uso de interfaces de segurança genéricas (GSS-API) para a comunicação entre objetos de serviço e serviços subjacentes, para que assim um objeto ***Vault*** não seja específico a uma única tecnologia de segurança.

Passo 4)	Torna as credenciais do cliente (incluindo seus atributos de segurança) disponíveis no servidor (colocá-las no atributo *received_credentials* do *Current*).

As credenciais do principal, que estão na lista ***own_credentials*** do objeto ***SecurityManager*** do cliente, precisam ser transferidas de forma transparente ao atributo ***received_credentials*** do ***Current*** do objeto destino. Isto ocorre via mecanismo de segurança subjacente (por exemplo, Kerberos e SSL). Entretanto, a informação transferida não é o objeto ***Credentials***, mas somente a informação que representa este principal. O objeto ***Vault***, no lado do servidor, usando a operação ***accept_security_context***, constrói o objeto ***Credentials*** que representa esta informação. Este irá validar essas credenciais, verificando sua confiança em relação à entidade que gerou o certificado para as mesmas. No fim do processo de ligação, essas credenciais estarão contidas no lado do servidor, no atributo ***received_credentials*** do ***Current***.

Tomando como exemplo o uso do Kerberos como suporte da tecnologia de segurança subjacente, um cliente precisa que o ***PrincipalAuthenticator*** crie o objeto ***Credentials*** para o Kerberos para que este possa realizar invocações sobre objetos servidores de aplicação. Supõe-se que o cliente tenha uma credencial Kerberos com o atributo *AccessId* com o valor de "maria@jacoweb". Este cliente, de posse de sua credencial Kerberos, deseja fazer uma invocação ao objeto B ("joão@jacoweb"). No Kerberos, o objeto ***Vault*** deve obter em um serviço de distribuição de *tickets* um *ticket* para que "maria@jacoweb" possa conversar com "joão@jacoweb". O ***Vault*** usa este *ticket* para gerar um *token* que pode ser entendido somente pelo "joão@jacoweb". Usando o protocolo SECIOP (seção 3.4.8), o objeto ***Vault*** (do cliente) envia este *token* em uma mensagem de estabelecimento de contexto. Se o ***Vault*** (do servidor) entende este *token*, fica fácil verificar quem enviou o *token*. Então, o ***Vault*** do lado do servidor, usando a operação ***accept_security_context***, constrói a credencial do cliente (objeto ***ReceivedCredentials***)

que contém, basicamente, o atributo *AccessId* com o valor "maria@jacoweb" (LISTA CORBA-SECURITY, 1999).

Usando o ***SSL*** como tecnologia de segurança, tem-se o CSI nível 0 (*Common Secure Interoperability Level 0*). Como este nível suporta somente políticas baseadas em identidade sem delegação, basta usar o **DN** (*Domain Name*) do certificado emitido pelo cliente para representar o principal com o atributo ***AccessId***. De acordo com a solução apresentada anteriormente, o objeto ***Vault*** deve, a partir da API da implementação SSL, analisar a cadeia de certificados do cliente para obter o **DN** do principal e construir o objeto ***Credentials***, no lado do servidor.

Para utilizar políticas baseadas nos atributos de privilégio definidos no processo de autenticação, a literatura sugere (OMG, 2000b), como solução ainda não padronizada pela OMG, o uso de um certificado especial que conterá os atributos de privilégio (PAC- *Privilege Attribute Certificate*). No caso do SSL, que ainda não suporta este certificado, pode-se definir no **DN** (*Domain Name*) um atributo de privilégio (por exemplo, *role*, *level* ou *group*), ao invés do atributo de identidade.

Resumindo, em ***tempo de binding***, o interceptador de chamada segura deve (Figura 3.12): obter as políticas necessárias para a invocação; executar o **procedimento de decisão;** estabelecer o contexto de segurança (objetos *SecurityContext*) e tornar as credenciais do cliente disponíveis no servidor.

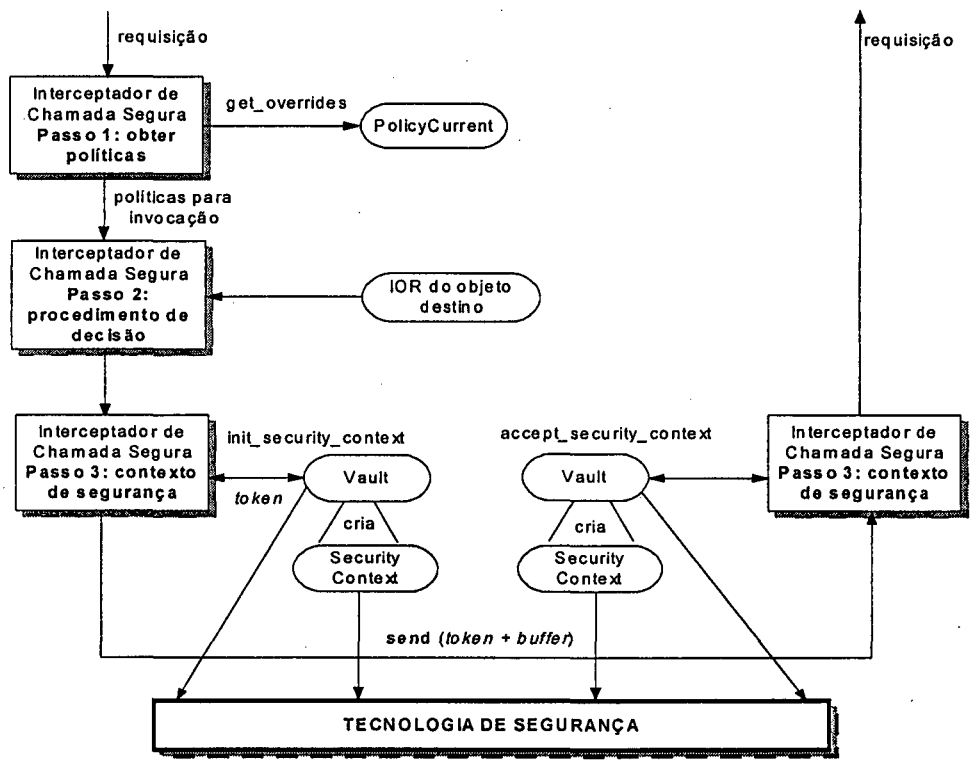


Figura 3.12 – Dinâmica do Interceptador de Chamada Segura durante o *binding*.

Message Protection Time

Depois de estabelecer o contexto de segurança, em **tempo de proteção de mensagem,** o interceptador de chamada segura é usado para proteger as mensagens, fornecendo integridade e/ou confidencialidade para as requisições e respostas, de acordo com os requisitos de qualidade de proteção especificados para a associação segura no objeto ***SecurityContext*** ativo.

No lado do cliente, usando o objeto ***ClientSecurityContext***, o método ***send_message*** do interceptador de chamada segura chama o método ***SecurityContext::protect_message*** para proteger as mensagens. Do lado do servidor, o método **receive_message** do interceptador chama o método **reclaim_message** do ***ServerSecurityContext***, para recuperar a mensagem.

Uma associação segura pode, em alguns ambientes, fornecer proteção através de mecanismos inerentes ao ambiente, como por exemplo, empregando uma camada de proteção abaixo da camada de mensagem do ORB. O ORB não necessita garantir sua própria proteção de mensagem quando este opera sobre uma camada de transporte segura (p.ex., o SSL).

Como o CORBAsec tem que atender a uma variedade de propósitos e níveis de proteção, as interfaces apresentadas na especificação permitem a escolha de algoritmos criptográficos para fornecer proteção. A qualidade da proteção depende da qualidade dos mecanismos subjacentes.

3.4.7 Auditoria

Serviços de auditoria permitem ao sistema gerenciar o monitoramento de eventos no ORB, através de registros dos detalhes de eventos relevantes à segurança do sistema, como por exemplo, tentativas de violação de segurança. Políticas de auditoria são usadas para restringir quais os tipos de eventos devem ser examinados e em quais circunstâncias.

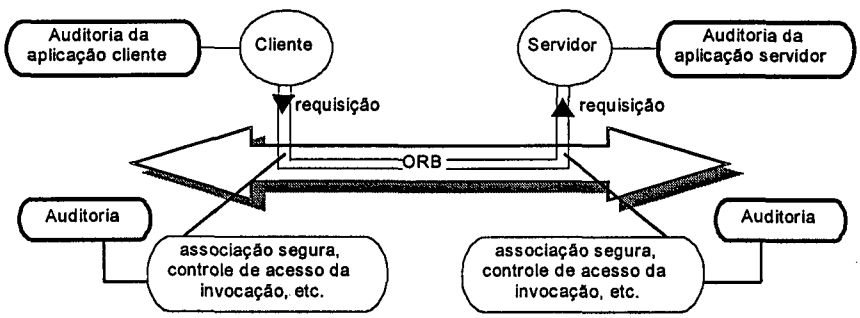


Figura 3.13 – Modelo do serviço de auditoria.

De acordo com a Figura 3.13, que apresenta o modelo do serviço de auditoria do modelo CORBAsec, têm-se duas categorias de políticas de auditoria:

- As **políticas de auditoria do sistema** que controlam quais eventos são registrados como resultado das atividades relevantes do sistema, como autenticação, mudanças de privilégio,

sucesso ou falha de invocações de objetos, e administração de políticas de segurança. Estas políticas de auditoria são aplicadas automaticamente pelo ORB para todas as aplicações (cientes ou não da segurança).

- As **políticas de auditoria da aplicação** que controla quais eventos são examinados pelas aplicações cientes da segurança.

Os eventos podem ser registrados sobre "rastros de auditoria" (*audit trails*) para depois serem analisados ou gerarem alarmes enviados aos administradores. A especificação CORBAsec apresenta como os registros de auditoria são gerados e escritos em canais de auditoria, mas não especifica como esses registros são filtrados depois, como os canais e os rastros da auditoria são mantidos seguros, e como os registros podem ser coletados e analisados.

Os requisitos de auditoria contemplam a geração dos dados de auditoria, a associação da identidade do usuário, a consulta dos dados de auditoria, a auditoria seletiva, as garantias de disponibilidade de dados de auditoria, a prevenção da perda dos dados de auditoria, e a ação no caso de perdas dos dados de auditoria.

Os objetos de serviço que participam das atividades de auditoria do CORBAsec são: *AuditDecision*, *AuditPolicy* e *AuditChannel* (BLAKLEY, 1999, OMG, 2000a). A cada evento considerado importante para o ambiente, a política de auditoria (objeto *AuditPolicy*) é consultada pelo objeto *AuditDecision* para decidir se um evento de auditoria deve ser gerado ou não. Caso seja necessário gerar o evento de auditoria, o objeto *AuditChannel* é localizado para registrar ou usar esses dados de alguma forma. Os dados que são enviados para o objeto *AuditChannel* registrar ou usar podem incluir (BLAKLEY, 1999, OMG, 2000a): o tipo do evento, as credenciais do requisitante do evento, o horário em que ocorreu o evento, os valores usados pelo objeto *AuditDecision* para decidir se a auditoria era necessária, e, ainda, outros dados específicos do evento.

3.4.8 Interoperabilidade e a Segurança

A interoperabilidade é um dos princípios de projeto mais importantes existente no CORBAsec (ALIREZA, LANG, *et al.*, 2000). O modelo de segurança CORBA na interoperabilidade entre ORBs segue as especificações CORBA e enfatiza uma tecnologia de segurança comum e um mesmo protocolo de interoperabilidade nos serviços subjacentes (LANG e SCHREINER, 2000). A especificação (OMG, 2000a) não garante interoperabilidade entre sistemas usando diferentes tecnologias de segurança.

Normalmente, uma referência de objeto IOR (*Interoperable Object Reference*) é *uma seqüência de profiles de protocolo* chamados *tagged profiles*. Um ***tagged profile*** é um mecanismo

de interoperabilidade que fornece aos clientes informações suficientes para possibilitar a comunicação com o servidor. Cada *tagged profile* suporta um ou mais protocolos e encapsula toda a informação básica que os protolocos suportados necessitam para identificar um objeto. Quaisquer *tagged profiles* armazenam informações suficientes para completar uma invocação usando qualquer um dos protocolos suportados; o conteúdo e a estrutura das entradas de um *tagged profile* é especificado caracterizando esses protocolos (OMG, 1999b). As referências de objeto (IORs) devem ter pelo menos um *tagged profile* por protocolo suportado (RUH, HERRON, *et al.*, 1999).

A especificação de interoperabilidade segura comum CSI (*Common Secure Interoperability Specification*), que faz parte da especificação geral do CORBAsec, identifica protocolos e tecnologias de segurança necessários para suportar a interoperabilidade segura, define os *tags* de segurança da IOR, ou seja, identifica os *tagged profiles* associados com cada um dos protocolos e tecnologias suportados que serão utilizados, e define a funcionalidade de segurança suportada por esses mecanismos (OMG, 1999c, 2000a).

A CSI introduz os seguintes protocolos e tecnologias de segurança como necessárias a interoperabilidade:

- *Secure Inter-ORB Protocol* (***SECIOP***) – usa os mecanismos **SPKM** (ADAMS, 1996), **GSS Kerberos** (KOHL e NEUMAN, 1993) ou **CSI-ECMA (SESAME)** (ECMA235, 1996).
- *Distributed Computing Environment Common Inter-ORB Protocol* (***DCE-CIOP***) – esse protocolo de transporte se baseia nos serviços de segurança DCE e no RPC autenticado do DCE (OPENGROUP, 2000).
- ***SSLIOP*** – define um protocolo de transporte 'plugável' (*pluggable*), que pode ser usado quando o *Secure Sockets Layer/Tranport Sockets Layer (SSL/TLS)* (DIERKS e ALLEN, 1999, FREIER, KARLTON, *et al.*, 1996) é a tecnologia de segurança subjacente.

O modelo de interoperabilidade proposto pela CSI é baseado no estabelecimento de um contexto de segurança, através da troca de um conjunto de *tokens* e credenciais que contêm informações de autenticação e da tecnologia de segurança que está sendo utilizada (LANG e SCHREINER, 2000, OMG, 2000a). Este modelo é fortemente vinculado ao SECIOP (Figura 3.14) e ao DCE-CIOP, que usam a GSS-API para acessar as respectivas tecnologias de segurança (RUH, HERRON, *et al.*, 1999). A introdução do SSL/TLS como possível tecnologia de segurança não prevê o uso de troca de *tokens* de segurança para o estabelecimento de um contexto de segurança (OMG, 2000b).

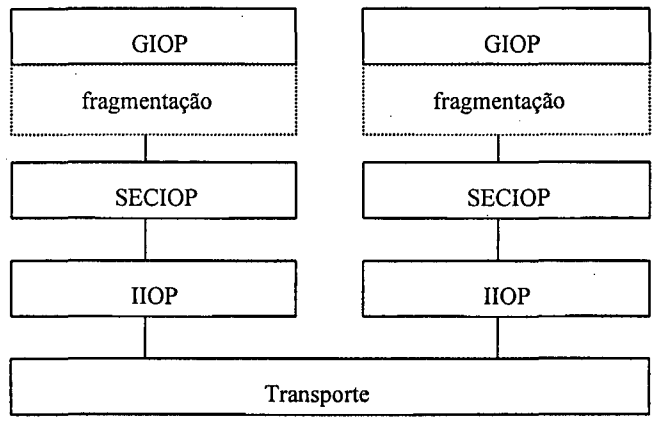


Figura 3.14 – Protocolo SECIOP.

Ulrich Lang (LANG e SCHREINER, 2000) afirma que quando um ORB opera sobre uma camada de transporte segura (por exemplo, o protocolo SSL), este não precisa fornecer sua própria proteção de mensagem. A integração do ORB com o SSL não é claramente especificada no CORBAsec. Como não há uma API (*Application Programming Interface*) padrão, geralmente, o ORB tem que manipular diretamente com uma implementação do protocolo SSL (*toolkit* SSL). Em (LISTA CORBA-SECURITY, 2000), Polar Humenn afirma que o contexto de segurança não precisa ser estabelecido com o SSL, já que os objetos ***Vault*** e ***SecurityContext*** estão baseados no protocolo SECIOP.

A CSI estabelece três níveis de funcionalidade de segurança:

- **Nível 0** - suporta política baseada em identidade, sem possibilidade de delegação, isto é, apenas a identidade do principal é transmitida a partir do objeto cliente para o servidor, sem possibilidade de delegação de direitos para outros objetos. O **SECIOP-SPKM** e o **SSLIOP-SSL/TLS** suportam este nível.
- **Nível 1** – suporta política baseada em identidade com delegação irrestrita, isto é, além de transmitir apenas a identidade do principal entre o objeto cliente e servidor, essa identidade pode ser delegada para quaisquer outros objetos em futuras invocações de objetos. O **SECIOP-GSS Kerberos** suporta este nível.
- **Nível 2** – suporta política baseada em identidades e privilégios com delegação controlada, isto é, nesse nível os atributos do principal, que são enviados a partir do objeto cliente para o servidor, podem incluir identidades de acesso e auditoria e uma série de privilégios, como papéis e grupos. A delegação desses atributos pode ser controlada de forma que esses atributos possam ser usados apenas em certas localizações. O **SECIOP-CSI-ECMA** e o **DCE-CIOP** suportam esse nível.

Na especificação CSI do CORBAsec atual (OMG, 2000a), os níveis de funcionalidade de segurança são fortemente vinculados aos mecanismos de segurança utilizados. A CSI está sofrendo um processo de revisão da especificação (OMG, 2000b, 2000c) para tratar dos requisitos do CORBAsec quanto a autenticação, delegação e privilégios interoperáveis. O modelo CSI que está sendo proposto (OMG, 1999c, 2000b, 2000c) é baseado em uma arquitetura de três camadas que estendem o modelo atual descrito em (OMG, 2000a): camada de autenticação, camada de proteção de mensagens e camada de autorização. Tem como objetivos: propor *tokens* padronizados a serem utilizados com quaisquer tecnologias de segurança (OMG, 2000c), fornecer várias escolhas de tecnologias (por exemplo, Kerberos, DCE, SSL/TLS, IPSEC, etc) para autenticação e proteção de mensagens (OMG, 2000b), e usar os interceptadores para implementar as camadas propostas (OMG, 1999c).

3.4.9 Pontos de Ameaças no CORBAsec

A Figura 3.15 mostra os sete pontos de ataques presentes em um ambiente que usa CORBAsec (CHIZMADIA, 2000). As ameaças e os mecanismos de segurança que impedem cada uma das ameaças são mostrados na Tabela 3.3. Outras ameaças presentes no ambiente são: o não cumprimento (*bypass)* dos controles de segurança, a falta ou perda da contabilização, e vulnerabilidades sem medidas de prevenção como a negação de serviço.

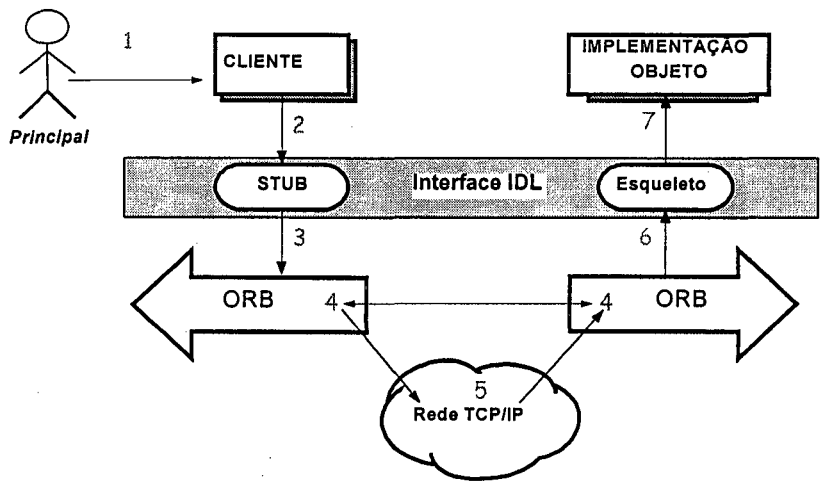


Figura 3.15 – Pontos de Ameaças no ambiente CORBAsec (CHIZMADIA, 2000).

Ponto de Ataque	Ameaças Presentes	Mecanismos de Segurança
1	▪ há a possibilidade do usuário se fazer passar por um usuário legítimo e autorizado; ▪ o usuário pode ter autorização em excesso.	▪ identificação do usuário, autenticação do usuário; ▪ controle de acesso do usuário.
2	▪ uso de uma implementação não desejável de um objeto; ▪ repudiação da requisição/resposta; ▪ descoberta de dados.	▪ controle de acesso a nível de aplicação; ▪ não-repudiação; ▪ registro de auditoria de segurança; ▪ proteção dos dados.
3	▪ aplicações (não conscientes da segurança) desprotegidas ; ▪ revelação involuntária da existência da máquina do cliente.	▪ controle de acesso na invocação dos objetos no lado do cliente; ▪ proteção dos dados.
4	▪ mascaramento de objetos; ▪ mascaramento do cliente; ▪ uso indevido das autorizações do usuário; ▪ alteração (*tampering*) da referência do objeto IOR; ▪ divulgação indevida (*disclosure*) dos conteúdos do *Request*; ▪ modificação/destruição dos conteúdos do *Request*.	▪ autenticação entre cliente e servidor; ▪ cifragem entre cliente e servidor; ▪ controles de delegação; ▪ registro de auditoria de segurança.
5	▪ intromissão para escuta (*eavesdropping*) na rede; ▪ alteração da mensagem (*tampering*); ▪ inabilidade para atravessar as fronteiras da rede (por exemplo, *firewalls*).	▪ cifragem na comunicação; ▪ passagem de IIOP pelos *firewalls*.
6	▪ aplicações (não conscientes da segurança) desprotegidas; ▪ número muito grande de interfaces e implementações de objetos a serem gerenciadas individualmente.	▪ controle de acesso na invocação dos objetos no lado do servidor; ▪ domínios de políticas de segurança.
7	▪ divulgação não autorizada de informações específicas para os clientes; ▪ repudiação da requisição/resposta; ▪ proteção de dados.	▪ controle de acesso a nível de aplicação; ▪ não-repudiação; ▪ registro de auditoria de segurança; ▪ proteção dos dados.

Tabela 3.3 – Ameaças no CORBAsec e os mecanismos de segurança relacionados.

3.5 *JaCoWeb*

O **Projeto *JaCoWeb Security* (*http://www.lcmi.ufsc.br/jacoweb*)** tem como objetivo o uso do modelo CORBA de segurança integrado com os modelos de segurança Java e Web, de modo a compor um esquema de autorização para aplicações distribuídas em redes de larga escala. Esse esquema está sendo desenvolvido com o objetivo de concretizar a implantação de políticas globais, o que representa um grande desafio, principalmente em redes de larga escala como a Internet (WESTPHALL e FRAGA, 1999a, 1999b, WESTPHALL, FRAGA, *et al.*, 2000).

No sentido de aproveitar as características desejáveis em sistemas distribuídos, como a disponibilidade de serviços e a transparência da distribuição, a segurança no *JaCoWeb* também é conduzida no sentido de algumas funções globais e outras locais, a exemplo do Delta-4 (DESWARTE, 1991, FRAGA, 1985). Na verdade, em sistemas distribuídos existem objetos

persistentes de grande granularidade (arquivos, *daemons* sobre o Unix etc), e objetos criados dinamicamente no sentido de preencher tarefas pontuais. Uma maneira adequada de implementar um esquema de autorização é localizar o nível global, tratando os acessos aos objetos persistentes e a autenticação, em um servidor de autorização. O nível correspondendo à segurança local, envolvendo o controle sobre objetos temporários ou localizados, é tratado pelos núcleos de segurança e TCBs locais.

No ambiente *JaCoWeb*, aplicações distribuídas são expressas na forma de clientes representados por códigos móveis ou *applets* Java, e de objetos servidores de aplicação, implementados como objetos distribuídos e visíveis via CORBA. O usuário, quando apresenta a URL do serviço desejado, provoca a carga no *browser* do *applet* residente na máquina servidora Web. A execução do código do *applet* pode conter chamadas a métodos remotos que são tratados pelo servidor CORBA na máquina servidora. O cliente (*applet* Java) interage com um servidor CORBA (objeto remoto) através de um ORB, em um modelo cliente/servidor tradicional[16].

O esquema de autorização define dois níveis de controle de segurança: o *nível global* e o *nível local*. Esses dois níveis são concretizados em objetos de serviço *COSS* e por núcleos de segurança e TCBs (*Trusted Computing Bases*), respectivamente. Os objetos de serviço, descritos na seção 3.4, concentram funções de identificação e autenticação de usuários, os controles de autorização no acesso de objetos de aplicação visíveis a nível global (visíveis via CORBA), e os controles de auditoria do sistema. Os núcleos de segurança e TCBs, presentes em cada máquina do sistema, validam os acessos aos recursos locais. A Figura 3.16 mostra os componentes principais que implementam os controles do nosso esquema.

[16] *Applets* clientes estão sujeitos às restrições de segurança impostas pelo *browsers* onde são executados. Estas restrições impedem que esses códigos possam ter acesso ao disco local ou façam conexões com sítios (*hosts*) diferentes de onde eles foram carregados. De alguma maneira isto impõe que os objetos servidores da aplicação estejam na mesma máquina do servidor Web a partir do qual o *applet* foi carregado. Algumas técnicas são oferecidas para permitir que os applets se libertem dessas restrições e possam se comunicar com objetos em um sistema distribuído sem limitação de localização. Neste trabalho, adotou-se a solução de *applets* assinados da plataforma Java™ (SUN, 1998, SUN, 1999). Na versão Java 1.2, a segurança é controlada pela política de segurança aplicada em tempo de execução. Portanto, pode-se configurar o arquivo de política para que privilégios sejam concedidos aos *applets*, livrando-os das restrições impostas.

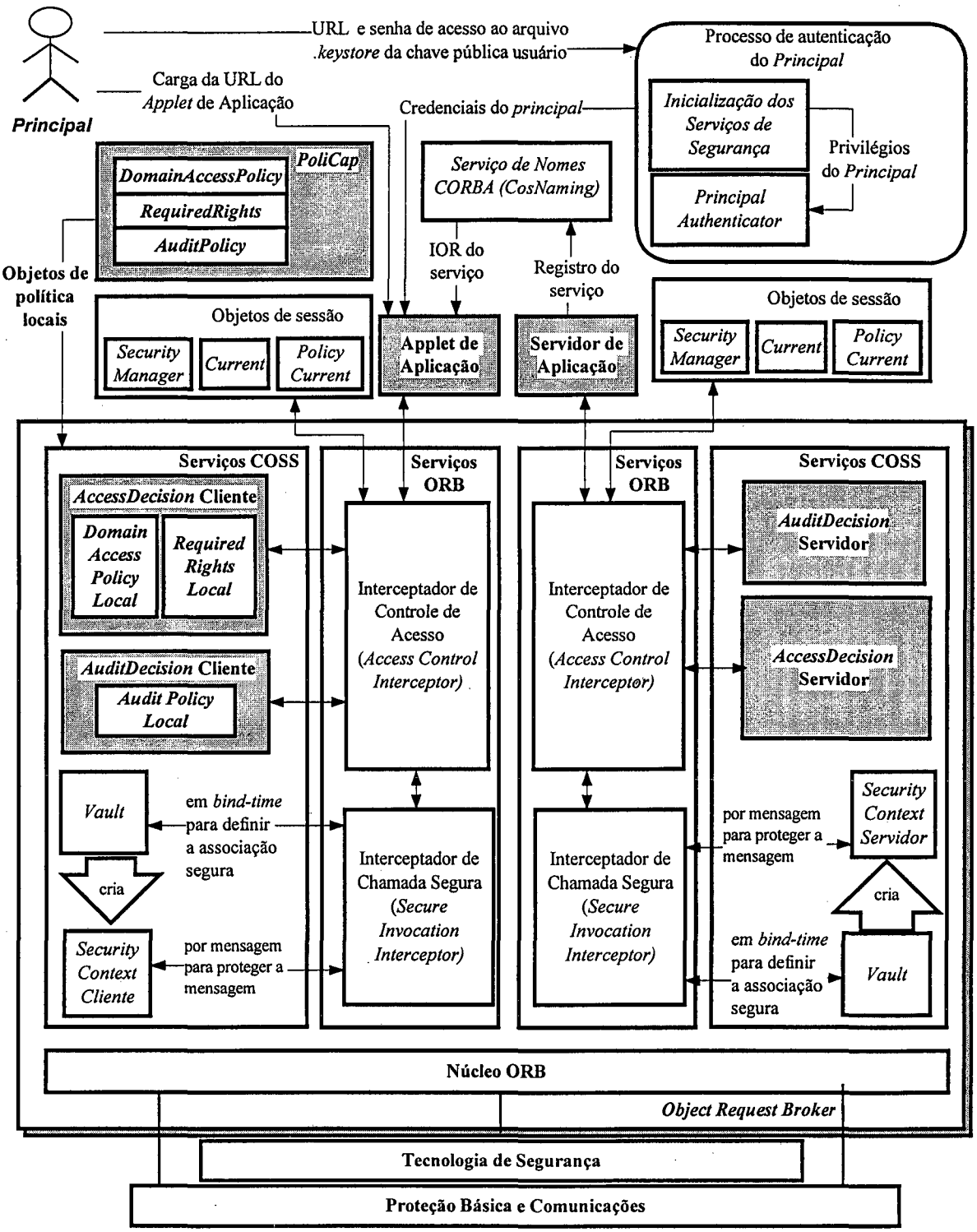


Figura 3.16 – Arquitetura do Esquema de Autorização *JaCoWeb*.

Os núcleos de segurança e *Trusted Computing Bases* validam os acessos locais dos *applets* de aplicação. Para implementar essa base confiável que valida os acessos locais, usamos o modelo de segurança Java com seus procedimentos de controle de acesso. O modelo de segurança Java, na sua versão 1.2, possui arquivos de política que identificam quais operações podem ser realizadas pelos códigos carregados (*applets*). Além disso, conta com o gerente de segurança (*Security Manager Class*) que realiza o controle de acesso, baseado na política concretizada através dos domínios de proteção. Para proteger a máquina hospedeira de um código móvel (*applet*), é necessário passar por uma fase de autenticação do código móvel, e durante a execução do mesmo, validar os acessos a recursos locais através do *Security Manager*. A autenticação serve para garantir a procedência do código. Os privilégios retornados (credenciais), quando da autenticação do usuário, determinam a construção do domínio de proteção associado ao *applet* de aplicação ativado pelo usuário. Esses domínios de proteção determinam a dependência desses acessos locais em relação às políticas globais.

O **Esquema de Autorização *JaCoWeb*** conta com mecanismos de **autenticação, controle de acesso, controles criptográficos** e mecanismos de **auditoria.**

3.5.1 Autenticação no JaCoWeb

Há três fases de autenticação presentes no *JaCoWeb* : a *autenticação do código móvel* ou *applet Java*, a *autenticação do principal* e a *autenticação da sessão.*

A **autenticação do *applet* Java**, que é carregado pelo *browser* e representa o cliente no ambiente considerado, tem como objetivo principal identificar se o código a ser executado foi assinado por um servidor considerado confiável pelo sistema. Para tanto o sistema mantém listas de servidores Web considerados confiáveis. Para realizar essa autenticação o ambiente usa os recursos de assinatura de *applets* provido pelo ambiente Java JDK 1.2 com a ferramenta *keytool*, *keystore* e *Policy* (OAKS, 1999, PISTOLA, RELLER, *et al.*, 1999, SUN, 1998).

A ***autenticação de usuários ou principais*** no esquema *JaCoWeb* é feita usando uma classe de inicialização dos serviços de segurança do ambiente, chamada *Security*, e também o objeto de serviço *PrincipalAuthenticator*. A classe *Security* e o *PrincipalAuthenticator* interagem segundo as trocas especificadas pelo CORBA para a identificação e autenticação de principais. O usuário, quando apresenta a URL do serviço desejado, deve fornecer também a URL e a senha de acesso ao arquivo *.keystore* que armazena a chave pública do usuário, seguindo o modelo de segurança Java.

A classe de inicialização *Security* localiza o certificado X.509v3 do usuário, armazenado neste arquivo *.keystore* do modelo de segurança Java. A partir desse certificado, a classe de inicialização obtém a identidade do usuário (*subjectDN* do certificado), realizando a identificação

no *JaCoWeb*, com o mapeamento entre identidade e seus respectivos atributos de privilégio no sistema (mapeamento esse que é mantido pelo administrador do sistema). Tendo os atributos de privilégio do principal no sistema, a classe de inicialização invoca o método *authenticate* do objeto *PrincipalAuthenticator*, fornecendo o mecanismo a ser utilizado na autenticação e os privilégios para que sejam criadas as credenciais do principal (objeto *Credentials*) no CORBAsec. A exemplo de uma das opções presentes no ORBasec (ADIRON, 2000), o método ou mecanismo a ser utilizado para a autenticação é o *SSLKeyStoreWithStorePass*. Nesse método, o usuário fornece a URL do arquivo *.keystore* para a localização da cadeia de certificados, bem como uma senha denominada *key_pass* que é usada para permitir acesso ao arquivo *.keystore*. O sistema mantém uma lista de entidades de certificação confiáveis para estabelecer a autenticação do principal. Uma vez verificada a identificação do usuário, estabelecendo as credenciais CORBA do cliente (objeto de serviço designado *Credentials*), o *applet* de aplicação (cliente CORBA) pode fazer requisições.

O mesmo certificado X.509v3, usado para a criação das credenciais, é utilizado para a autenticação mútua entre cliente e servidor (**autenticação da sessão** do modelo CORBAsec), que é feita, durante o estabelecimento da associação segura do CORBAsec, pela tecnologia de segurança usada em nosso protótipo: o SSL. Neste caso, é utilizado todo o suporte criptográfico fornecido pelo SSL e IAIK (ver capítulo 4). Um padrão para estrutura de chaves públicas, a ser utilizada em ambientes CORBA, está sendo proposta pela OMG (OMG, 2000d).

3.5.2 Controle de Acesso

O *applet* de aplicação (Figura 3.16), depois de autenticado, interage com o serviço de nomes CORBA (*CosNaming*) (BROSE, 1997, OMG, 1999b), para obter, a partir do nome do objeto, a referência ou IOR (*Interoperable Object Reference*) do objeto servidor de aplicação. Isso deve permitir a ligação com o objeto servidor. As chamadas executadas por esse *applet* de aplicação sobre um servidor remoto da aplicação estão sujeitas a dois níveis de controle de acesso. No nível mais alto de controle de acesso, a verificação primeiramente ocorre em tempo de ligação e, uma vez validada a requisição, o serviço de política de autorização do domínio de segurança do objeto servidor fornece versões apropriadas dos objetos de política *DomainAccessPolicy*, *AuditPolicy* e *RequiredRights*, que vão atuar nas interações cliente/servidor em tempo de decisão de acesso. Ainda em tempo de decisão de acesso, a partir desta verificação de alto nível no lado do cliente, no *JaCoWeb* são geradas *capabilities* (competências) que, por sua vez, são validadas na máquina do objeto servidor remoto, completando assim o segundo nível de controle de acesso no nosso esquema de autorização (nível local).

Para montar esses dois níveis de controle de acesso, utilizamos os dois níveis de interceptação definidos no modelo CORBA de segurança e seus objetos de serviço (COSS). A

interceptação de alto nível (*Controle de Acesso*), no lado do cliente (*applet* de aplicação), desvia para o objeto de serviço *AccessDecision*, que obtém *em tempo de ligação*, a partir do serviço de política *PoliCap* (detalhado no capítulo 4), os objetos locais (*DomainAccessPolicy*, *AuditPolicy* e *RequiredRights*), a serem usados nas verificações do mecanismo de *capabilities em tempo de decisão de acesso*. No lado do servidor da aplicação, a interceptação de alto nível é usada para validar a *capability* recebida com a requisição da invocação. As políticas **discricionárias** são implementadas conforme o modelo CORBAsec, e são descritas no capítulo 4. Políticas **obrigatórias** do *JaCoWeb* são apresentadas no capítulo 5.

A interceptação de baixo nível (*Chamada Segura*) do modelo CORBA, em ambos os lados (cliente e servidor), é usada para proteger em uma associação segura a mensagem que transporta a requisição com a *capability*. A representação de privilégios (ou direitos), na forma de *capabilities* no esquema de autorização, é detalhada no capítulo 4.

3.5.3 Controles Criptográficos

Além dos controles citados acima, os controles criptográficos também são necessários no esquema, e são definidos também na forma de objetos de serviço CORBA que usam a tecnologia de segurança residente sob o ORB.

Durante o desenvolvimento do CORBAsec, o SESAME era o mecanismo de segurança mais poderoso do momento, e portanto, os conceitos básicos desse mecanismo são encontrados no CORBAsec. Entretanto, o SSL tornou-se mais amplamente utilizado que o SESAME. Dessa forma, o CORBAsec sobre o SSL não é tão poderoso como o SESAME, e muitas características do CORBAsec não se aplicam no uso do SSL (ALIREZA, LANG, *et al.*, 2000). O trabalho da OMG neste sentido foi apresentado na seção 3.4.8.

Para implementar os controles criptográficos no *JaCoWeb*, o SSL (*Secure Socket Layer*) foi usado como tecnologia de segurança subjacente. A integração do protocolo 'plugável' (*pluggable*) SSL no ORB foi baseada na abordagem de objetos de serviço do CORBA, que não altera as características e funcionalidades originais do ORB, mas permite executar tanto requisições convencionais quanto seguras (WANGHAM, LUNG, *et al.*, 2000). O *framework* de integração do ORB/CORBA e SSL consiste em encapsular um pacote (conjunto de classes) pronto que implementa o protocolo SSL na forma de um objeto de serviço. O capítulo 4 descreve melhor essa integração da tecnologia de segurança SSL no esquema *JaCoWeb*.

3.5.4 Auditoria

A geração de dados de auditoria é feita via arquivos de *logs* e implementada pela política de auditoria (objeto *AuditPolicy*) definida no modelo CORBAsec. Essa política é definida pelo administrador do sistema e armazenada no serviço de políticas *PoliCap,* sendo carregada em ambos os lados em tempo de ligação entre objetos cliente e servidor.

Aplicações administrativas provêem acesso aos dados de auditoria apenas para o administrador do sistema, que pode revisar e selecionar dados de interesse.

A auditoria no *JaCoWeb* usa os objetos de serviço do CORBAsec *AuditDecision*, *AuditPolicy* e *AuditChannel* (BLAKLEY, 1999, OMG, 2000a). A cada evento considerado importante para o sistema, a política de auditoria (objeto *AuditPolicy*) é consultada pelo objeto *AuditDecision* para decidir se um evento de auditoria deve ser gerado ou não. Caso seja necessário gerar o evento de auditoria, o objeto *AuditChannel* é localizado para registrar ou usar esses dados de alguma forma. Os dados, que são enviados para o objeto *AuditChannel* registrar ou usar, podem incluir (BLAKLEY, 1999, OMG, 2000a): o tipo do evento, as credenciais do requisitante do evento, o horário em que ocorreu o evento, os valores usados pelo objeto *AuditDecision* para decidir se a auditoria era necessária, e, ainda, outros dados específicos do evento.

Quando os objetos envolvidos na auditoria são usados para implementar políticas de auditoria no sistema, garantidas no nível do ORB, existem funções que são executadas *em tempo de binding* e em *tempo de decisão de acesso.*

Em *tempo de binding*, o interceptador de controle de acesso no lado do cliente invoca o método *audit_needed* do objeto *AuditDecision* para verificar se a geração de um evento de auditoria é necessária. Nesse momento, foi definido na política *AuditPolicy* que o evento relevante a ser verificado é a entrada do usuário no sistema (evento do tipo *AuditSessionAuth*, associado à autenticação do usuário). Depois de um certo tempo, o administrador é responsável por eliminar dados desnecessários dos arquivos de *log* gerados.

Em *access decision time* ou em tempo de controle de acesso, em ambos lados, cliente e servidor, o interceptador de controle de acesso invoca o objeto *AuditDecision* para as respectivas verificações de possibilidade de geração de eventos de auditoria. No lado do cliente, apenas o caso onde a autorização de acesso ao objeto servidor seja negada pelo objeto *AccessDecision* pode provocar um destes eventos, ou seja, o *principal* não é autorizado ao acesso desejado. Nessa situação, o objeto *AuditDecision* é invocado a fim de registrar o resultado negativo da autorização (evento do tipo *AuditAuthorization*). No lado do servidor, o objeto *AuditDecision* é chamado caso a verificação da *capability* apresente alguma irregularidade, sendo necessário o registro desse evento (evento do tipo *AuditAuthorization*).

3.6 *JaCoWeb* em Sistemas Distribuídos de Larga Escala

Em redes de larga escala, como a Internet, é difícil implementar serviços de autenticação e autorização (NEUMAN, 1994). As funções de nível global nesses sistemas podem se limitar a um serviço de autenticação normalmente na forma hierarquizada, como por exemplo o padrão proposto no X.509. Nessa abordagem, introduzida como um padrão ITU-T (ITU-TX509, 1993), o serviço de autenticação é particionado no sentido de aumentar a sua aplicabilidade em redes de larga escala como a Internet. Diferentes autoridades certificadoras (CA – *Certification Authority*) são estruturadas na forma de uma árvore. Um cliente particular é registrado em um desses CAs. O acesso de um cliente a objetos de um servidor registrado em outro CA implica na interação entre os CAs do cliente e do servidor no processo de autenticação.

O projeto *JaCoWeb Security* necessita portanto da proposição de uma estrutura para realizar a gestão da autorização em redes de larga escala como a Internet. O uso de tal estratégia de autorização em redes de larga escala passa, necessariamente, pela conexão de vários servidores de autorização. Para redes compostas de milhares de computadores, fica impossível a gestão de todos os objetos e usuários com um único servidor de autorização. Assim, para gerenciar de maneira centralizada nesses sistemas, é necessário dividir os usuários e objetos em domínios de segurança. Cada domínio deve ter um servidor de autorização. Esse servidor de autorização atua com a colaboração de um servidor de nomes no domínio. O servidor de nomes armazena informações relativas aos usuários e objetos do domínio. Esse servidor de nomes pode ser facilmente estendido e conectado a outros servidores de nomes de outros domínios. Para realizar essa conexão entre servidores de nomes, o *JaCoWeb* pretende usar o LDAP.

Normalmente, aplicações CORBA usam o serviço de nomes CORBA (*CosNaming*) para armazenar e recuperar referências de objetos. Os serviços de nomes CORBA podem ser estruturados de forma federada. A federação de serviço de nomes CORBA consiste na conexão de vários contextos de nomes. Entretanto, mecanismos de busca nesses serviços de nomes CORBA federados que realizem pesquisas em domínios diferentes não são padronizados, e, portanto, são dependentes de implementação.

Entretanto, no sentido de possibilitar essa busca automática de referências de objetos CORBA, existe um esforço que considera o armazenamento de objetos CORBA usando o diretório LDAP (RYAN e SELIGMAN, 1999). A motivação desse trabalho é definir uma forma comum de armazenar e recuperar referências de objetos CORBA a partir de um serviço de diretório[17] LDAP. Usando essa forma comum, qualquer aplicação CORBA que tem necessidade de armazenar e

[17] Um serviço de diretório pode ser considerado um tipo específico de serviço de nomes.

recuperar referências de objetos em um diretório pode fazê-lo de forma interoperável (JOHNER, BROWN, *et al.*, 1998, RYAN e SELIGMAN, 1999).

O LDAP (*Lightweight Directory Access Protocol*) é uma implementação do serviço de diretórios baseado no X.500 que pode não armazenar toda a árvore que compõe a estrutura de diretórios global. Esse servidor de nomes pode ser dividido em partes e ficar fisicamente armazenado em locais diferentes. As conexões entre os vários servidores LDAP pode ser feita usando os mecanismos de *sufixo* e *referrals* (WAHL, HOWES, *et al.*, 1997), já definidos na seção 2.6.3.4.

O projeto *JaCoWeb* pretende fazer uso de vários servidores LDAP distribuídos, sendo que cada um destes servidores é responsável pelo armazenamento dos objetos dos domínios de política locais, e está conectado aos outros servidores na forma de uma hierarquia, usando os mecanismos de *sufixo* e *referrals* próprios do LDAP (ver seção 2.6.3.4). Dessa forma, cada domínio de segurança no *JaCoWeb* possui um serviço de política *PoliCap* atuando em cooperação com um servidor LDAP. O *PoliCap* realiza o papel de servidor de autorização do domínio e o servidor LDAP atua como o serviço de nomes do domínio, estando conectado aos outros servidores LDAP, compondo a estrutura necessária para o armazenamento de todos os objetos de aplicação e objetos de serviço que participam do Esquema de Autorização *JaCoWeb* em larga escala.

Usando o LDAP, pode-se usar os serviços de proteção como a autenticação do SSL, para o acesso a referências de objetos do CORBAsec, como o *DomainAccessPolicy*, *AuditPolicy*, *RequiredRights* e outros, que são acessadas somente por objetos de serviço autorizados pelo esquema de autorização. No momento, o *JaCoWeb* usa o serviço de nomes CORBA em um domínio único sendo que o uso do LDAP é vislumbrado como perspectiva futura.

3.7 Experiências de ORBs Seguros - Acadêmicos e Comerciais

Uma experiência acadêmica relacionada ao uso do modelo CORBAsec foi desenvolvida pela Universidade de Illinois (EUA): o projeto *Nephilim*. Este projeto apresenta uma implementação em Java do modelo de segurança CORBA: *Java Implementation of CORBA Security Services* (NEPHILIM, 1999). Este trabalho fornece um pacote chamado JGSS[18] para suportar autenticação baseada no SESAME e protocolos de delegação. Este pacote fornece a interoperabilidade CSI-ECMA (baseado no SESAME e no mecanismo ECMA GSS-API), suportando o CSI nível 2. Usando interfaces padrão de interceptadores, esta experiência visa implementar controle de acesso no nível de ORB e objetos de delegação e auditoria segura, conforme especificado no serviço de segurança CORBA no nível 2 de funcionalidade de segurança.

[18]Pacote Java que implementa uma interface de segurança genérica (GSS-API).

O projeto *JacORBoverSSL*[19], desenvolvido por Andre Benvenuti, é uma implementação em Java que visa integrar um protocolo SSL ao JacORB (BROSE, 1997). Esta implementação do IIOP sobre o SSL foi realizada no núcleo do ORB.

Além das experiências acadêmicas citadas acima, existem alguns produtos comerciais que fornecem ORBs seguros. Podem ser destacados: *C5Sec* (CONCEPT5, 1998), *OrbixSecurity* (IONA, 1999), *Jbroker* (SILVERSTREAM, 2000), *ORBAsec SL2* (ADIRON, 2000), *VoyagerSecurity* (OBJECTSPACE, 1999) e *SecureBroker* (PROMIA, 1999). Uma breve análise desses ORBs é apresentada a seguir.

O ***C5Sec***, da empresa Concept5 Technologies Inc., implementa a funcionalidade de nível 2 do CORBAsec e foi integrado ao produto *OrbixSecurity* 3.0. Analisando a documentação do ***OrbixSecurity*** (IONA, 1999), verifica-se que a implementação deste ORB seguro atende ao nível 2 de funcionalidade de segurança do CORBAsec. Três mecanismos de autenticação de principais são suportados: *user ID* e *password login*; *login* usando *Security Dynamics SecurID*; e esquema de autenticação baseado em chave pública. Este produto fornece um controle de acesso distribuído, incluindo um controle a nível de aplicação com o auxílio de Certificados de Atributos de Privilégios (PACs) usando o padrão X500 e de certificados SSL (para os atributos de identidade). A segurança da comunicação (integridade e confidencialidade) é garantida pelo protocolo SSL, usado como tecnologia de segurança subjacente. Auditorias do processo de autenticação, estabelecimento da comunicação e invocação de objetos podem ser realizadas por este ORB seguro. O *OrbixSecurity* implementa as interfaces de administração de segurança conforme requeridas pelo nível 2 do CORBAsec. Constatou-se que o *OrbixSecurity* segue a maioria das interfaces definidas no CORBAsec, apesar de usar algumas funcionalidades que ainda não foram adicionadas na especificação, mas que possivelmente serão (por exemplo, o uso de PAC, gerência de domínios e algumas modificações no processo de auditoria). Além disso, o *OrbixSecurity* não implementa interceptadores padrão. Os mecanismos de desvio para controle de acesso são implementados através de filtros (*filters*), presentes no *Orbix*. Outro tipo de interceptador presente no *OrbixSecurity* é o *transformador* (*transformer*) que, diferentemente do filtro, pode operar sobre os pedidos. Neste produto, *transformadores* são usados na interceptação de baixo nível.

O **Jbroker** da SilverStream Inc. fornece o suporte para o uso do IIOP sobre SSL como definido na especificação *CORBAsec*. Entretanto, os processos de autenticação e controle de acesso não seguem os modelos definidos na especificação do serviço de segurança CORBA.

O **ORBAsec SL2** (ADIRON, 2000) é fornecido pela Adiron e cumpre parcialmente o modelo do CORBASec. Ele usa o Kerberos e o SSL como tecnologias de segurança. Autenticação

[19] Em breve, esta implementação será incorporada e distribuída no JacORB v1.0.

e associações seguras são fornecidas por esta implementação. O ORBAsec SL2 não atende totalmente ao nível 2 de funcionalidade de segurança CORBA; os objetos ***RequiredRights***, ***DomainAccessPolicy***, ***AccessDecision*** e ***AuditDecision*** não são usados nesta implementação.

O ORB seguro da Object Space (**VoyagerSecurity**) (OBJECTSPACE, 1999) fornece um *framework* de segurança que garante a integridade e confidencialidade dos dados com o uso do protocolo SSL. A integração desta tecnologia, bem como os serviços de autenticação e controle de acesso, não estão de acordo com os modelos e interfaces definidos no *CORBASec*.

O **SecureBroker** (PROMIA, 1999) ainda está em fase de desenvolvimento, e somente a integração com o SSL está disponível (SecureBrokerSSL). O *SecureBroker CORBA Security 2.0 Java version* empregará o Kerberos/SESAME, que utilizará o JacORB (ORB desenvolvido na Universidade de Berlim) (BROSE, 1997) e fornecerá comunicação cifrada, controle de acesso e auditoria.

O quadro comparativo na Tabela 3.4 apresenta as características dos produtos comerciais descritos acima.

ORBs SEGUROS	CARACTERÍSTICAS DE ACORDO COM A ESPECIFICAÇÃO CORBAsec				
	Autenticação	Associação segura	Controle de acesso	Interceptadores padrão	Nível de funcionalidade
OrbixSecurity	Mecanismos de autenticação subjacentes	Garantida pelo SSL	A nível de aplicação e a nível de middleware	Não implementa	Nível 2
Jbroker	Não segue a especificação	Garantida pelo SSL	Não segue a especificação	?	?
ORBAsec SL2	Kerberos e SSL	Garantida pelo Kerberos e SSL	Não fornece	Não implementa	Nível 2 (parcialmente)
VoyagerSecurity	Não fornece	Não segue a especificação	Não fornece	?	Não segue a especificação
SecureBroker (em desenvolvimento)	Kerberos/ SESAME	Kerberos / SESAME e SSL	A nível de aplicação e a nível de middleware	Implementa	Nível 2

Obs: Usou-se o símbolo ? quando as definições constantes da bibliografia consultada não possibilitaram a verificação da compatibilidade da característica com a especificação CORBAsec

Tabela 3.4 – Quadro Comparativo de ORBs seguros (WANGHAM, 2000).

3.8 Considerações sobre o CORBAsec e as Experiências de ORBs seguros

As especificações CORBAsec não definem esquemas de autorização ou técnicas específicas, antes de tudo criam uma estrutura geral que pode ser adaptada, em diferentes esquemas de autorização, para a implementação de diversas técnicas e mecanismos de segurança.

O CORBAsec foi o primeiro sistema de segurança para *middleware* orientado a objetos. A especificação não está totalmente amadurecida. Existem ainda vários problemas, desafios e fraquezas presentes no CORBAsec que devem ser tratados. Muitos problemas existentes no CORBAsec não são propriamente problemas isolados dessa especificação e sim, são dificuldades associadas à segurança de sistemas de objetos distribuídos de forma geral (ALIREZA, LANG, *et al.*, 2000).

Entre alguns dos desafios, problemas e fraquezas que podem ser citados e que estão em fase de melhorias, encontram-se os seguintes (ALIREZA, LANG, *et al.*, 2000, LANG e SCHREINER, 2000):

- **Integração do SSL**: O CORBAsec foi inicialmente desenvolvido para aplicações estáticas em ambientes restritos. A integração do SSL (soluções proprietárias), por volta de 1997, foi uma das primeiras tentativas de adaptar o CORBAsec para a Internet. Entretanto, a integração de protocolos 'plugáveis' (*pluggable*) não está totalmente padronizada pela OMG, sendo que várias questões sobre esses tipos de protocolos e o impacto na interoperabilidade estão sendo discutidos (ALIREZA, LANG, *et al.*, 2000, OMG, 2000b, 2000c).
- **Autenticação e Autorização**: Os atributos pré-definidos pela OMG são limitados e não descrevem todas as características de um principal.
- **Gerência das Políticas de Segurança**: A interface de administração *SecurityAdmin* do CORBAsec fornece uma definição rudimentar da forma como os objetos de política podem ser acessados e usados. A versão atual da especificação não trata muitas questões importantes como procedimentos para gerenciar membros dos domínios e para incluir ou remover políticas (ALIREZA, LANG, *et al.*, 2000, OMG, 2000a , 2000f).
- **Integração Portável**: A integração portável do CORBAsec em um ORB pode contar com três alternativas para atingir esta portabilidade (ALIREZA, LANG, *et al.*, 2000): através da modificação do próprio ORB, usando protocolos plugáveis, ou usando os interceptadores. A primeira abordagem requer o código fonte do ORB, o que não é disponível em muitos produtos comerciais, e, também, não segue o princípio de implementação de serviços adicionais como objeto de serviço da OMG (OMG, 1999b). Protocolos plugáveis representam uma abstração dos mecanismos de transporte do ORB. Entretanto, como a integração de protocolos plugáveis

ainda não está padronizada, cada ORB implementa interfaces proprietárias. Os interceptadores representam um papel importante com relação `a integração do CORBAsec em um ORB, especialmente se o ORB não fornece código fonte disponível. Entretanto, ainda não existe um padrão para interceptadores portáveis na OMG.

- **Dependências Externas**: O CORBAsec não pode ser visto de forma isolada, pois ele depende de serviços externos como o serviço de nomes (*Naming Service*), serviços de persistência de objetos (*Persistent Object Service*), dentre outros. No CORBAsec, alguns conceitos são um tanto novos e, portanto, interfaces estáveis a componentes externos podem não ser disponíveis imediatamente. Não é fácil determinar como novas propostas podem se encaixar na arquitetura do CORBAsec, tais como o *draft* de infraestrutura de chaves pública CORBA PKI (OMG, 2000d) e o *draft* para *firewall* CORBA (OMG, 2000e).
- **Tendências da OMG**: A OMG está trabalhando no sentido de ter subconjuntos do CORBAsec para domínios de aplicação específicos, tais como intranets de empresas, comércio eletrônico, telecomunicações e área de saúde. As discussões atuais na OMG quanto a adoção de um *CORBAsec Light* (CORBAsec leve e mínimo), indica que esse será o caminho certo a ser seguido.

Em relação às experiências de ORBs seguros, tomando como base o quadro da Tabela 3.4 e a literatura que descreve estes ORBs citados na seção 3.6, observamos que a maioria dos ORBs não segue a especificação completamente, e a integração do SSL aos ORBs, a autenticação, a autorização e a gerência das políticas de segurança seguem **soluções proprietárias** em vista dos **problemas** apresentados pela especificação do **CORBAsec**, e, também, **pela falta de experiência** na implementação deste padrão tão complexo e abrangente.

3.9 Conclusões do capítulo

Esse capítulo apresentou as vantagens e motivos do sucesso no uso da combinação das plataformas Java, CORBA e Web para a programação distribuída em redes de larga escala como a Internet. Essas novas abordagens introduzem também novos problemas de segurança e também implicam na necessidade de novos conceitos e modelos de segurança diante desse novo quadro.

Neste sentido, o ***Projeto JaCoWeb Security*** propõe um esquema de autorização para a programação distribuída em redes de larga escala como a Internet. O *JaCoWeb* integra os sistemas Java de códigos móveis e Web, com seus serviços de segurança, aos serviços de comunicação e segurança definidos pelo CORBAsec, compondo uma infraestrutura muito poderosa para a definição do nosso esquema de autorização.

O CORBA, por contar com um conjunto rico de características, como a reusabilidade, portabilidade e interoperabilidade de *softwares*, orientados a objetos, é atualmente um padrão importante e respeitado para integração de aplicações distribuídas. Além disso, a plataforma CORBA está se constituindo um padrão *de facto* para programação distribuída na Internet. Entretanto, algumas aplicações de sistemas de larga escala exigem uma característica a mais: a segurança. Em resposta a esta necessidade, a OMG adicionou o serviço de segurança CORBA à especificação *CORBAservices*.

A especificação de segurança do CORBA (OMG, 2000a) é a maior e mais complexa especificação produzida pela OMG, e abrange uma ampla série de requisitos de segurança (BLAKLEY, 1999). Algumas abstrações dessa especificação podem levar a uma interpretação errônea dos objetos de segurança CORBA, bem como dos objetos de segurança do ORB ou das funcionalidades de segurança do modelo CORBAsec. A compreensão sobre o seu funcionamento se baseia no entendimento, no nosso ponto de vista, de vários conceitos, como, por exemplo, os interceptadores e a dinâmica do uso dos objetos do CORBAsec. Essa dinâmica não é claramente descrita na literatura (BLAKLEY, 1999, OMG, 2000a) e requer um esforço considerável para o seu entendimento. Neste sentido, o texto apresentado nesse capítulo contribui de forma bastante significativa.

O CORBAsec está em constante evolução, sendo que várias extensões da especificação do CORBAsec estão em andamento atualmente (OMG, 1999c, 2000b, 2000c, 2000d, 2000e, 2000f).

Para a concepção de esquemas de autorização, vários pontos não são claramente abordados nas especificações CORBA. Por exemplo, nada é especificado em relação à abordagem de implementação das funções de segurança: se deve ser adotado uma abordagem centralizada ou descentralizada.

Poucos são os produtos comerciais existentes que implementam o serviço CORBA de segurança conforme o padrão. Isso pode ser devido à complexidade desse serviço, e, também, pelo pouco tempo de experiência na implementação dessa especificação.

Um dos objetivos do Projeto *JaCoWeb* é estar em conformidade com o modelo CORBAsec. Nesse aspecto, se preocupa em usar os padrões até então definidos para o modelo, aprimorando e estendendo, de certa forma, aspectos ainda não cobertos pela especificação como a gerência das políticas de segurança (capítulo 4) e o uso de políticas obrigatórias na autorização (capítulo 5).

Capítulo 4

ESQUEMA DE AUTORIZAÇÃO DISCRICIONÁRIO

4.1 Introdução

O controle de acesso em sistemas de larga escala é problemático. Se raciocinarmos em termos de objetos distribuídos, os objetos são constantemente adicionados, removidos e modificados nestes ambientes, definindo um número sem limites de componentes. Em sistemas geograficamente muito grandes normalmente existem diferentes domínios de políticas de segurança, o que contorna o problema da escalabilidade porém dificulta a interação entre as diferentes administrações.

O uso e implementação do padrão CORBAsec visa preencher essas necessidades de escalabilidade e interoperabilidade. O padrão CORBA de segurança é muito extenso (como pode ser observado no capítulo 3) e há pouca bibliografia que descreva seus principais aspectos de forma clara e sucinta, demonstrando seu uso em aplicações práticas. Atualmente, o número de empresas que desenvolve ORBs seguros ainda é pequeno.

Um dos objetivos do Projeto *JaCoWeb* é estar em conformidade com o modelo CORBAsec. Nesse aspecto, se preocupa em usar os padrões até então definidos para o modelo, aprimorando e estendendo, de certa forma, aspectos ainda não cobertos pela especificação.

Este capítulo, num primeiro momento, mostra a utilidade do Esquema de Autorização *JaCoWeb* na implantação de políticas discricionárias (WESTPHAL e FRAGA, 1999a, 1999b, WESTPHALL, FRAGA, *et al.*, 2000). Neste sentido, em seqüência são descritos:

- O serviço de política *PoliCap*: tem como objetivo possibilitar o gerenciamento a nível global de objetos de política em um domínio de aplicações de objetos distribuídos. As nossas propostas suprem as carências identificadas do *CORBAsec* quanto ao gerenciamento de objetos de política, e foram desenvolvidas no sentido de atuarem no esquema de autorização do modelo do projeto *JaCoWeb Security*. O ***PoliCap*** constitui o primeiro nível de verificação de controle de acesso no esquema *JaCoWeb* (seção 4.2).
- Um segundo nível de controle de acesso baseado em mecanismo de ***capabilities***: esse controle é realizado localmente, em cada nó servidor de aplicação, em tempo de execução da aplicação, refletindo as políticas do *PoliCap*. O mecanismo de *capabilities* é montado usando as

abstrações do próprio *CORBA,* provendo redução no tráfego de rede em caso de negações de acesso ao objeto servidor (seção 4.3). Este segundo nível junto com as verificações do *PoliCap* determinam mecanismos de controle em ambos os lados - no cliente e servidor - durante uma invocação de método.

Numa segunda etapa neste capítulo são descritas as implementações referentes a estes controles discricionários no *JaCoWeb* (pacote *JaCoWebSecurity*). Este conjunto de classes implementa os objetos de segurança identificados no modelo CORBAsec, e conta com as seguintes características de segurança (WANGHAM, 2000, WANGHAM, LUNG, *et al.*, 2000, WESTPHALL, FRAGA, *et al.*, 2000):

- **autenticação** do código a ser executado: esta autenticação está baseada na assinatura digital do *applet* de aplicação;
- **controle de acesso discricionário**: os controles discricionários são executados de forma transparente, a partir do ORB.

No sentido de avaliar os conceitos do *JaCoWeb*, nos seus controles discricionários, foi desenvolvida uma aplicação de *Internet Banking*.

4.2 *PoliCap* – Um Serviço de Política para o CORBAsec

Em sistemas de objetos distribuídos, existem várias questões que devem ser analisadas no momento da implantação de políticas de autorização: se as políticas serão **globais, locais, distribuídas** ou **dinâmicas** (LANG, 1999).

A maioria dos sistemas distribuídos em empresas são administrados de forma centralizada. Esse fato torna possível integrar aspectos relacionados com uma **política global** às definições de **políticas locais**. O nosso objetivo aqui é mostrar que é possível implementar este estilo de integração de políticas usando os conceitos CORBA. O esquema de autorização que resulta da implantação destas políticas fica particionado, envolvendo controles em diferentes locais.

Políticas de autorização distribuídas podem ser **dinâmicas** quando modificadas de acordo com os seus ambientes e com as decisões de acesso prévias.

O **serviço de política *PoliCap***, proposto neste trabalho, tem como **objetivo principal** possibilitar o gerenciamento de objetos de política de um domínio de uma aplicação de objetos distribuídos, suprindo a carência nas especificações do CORBAsec quanto ao gerenciamento de objetos de política (WESTPHALL, FRAGA, *et al.*, 2000). Além disso, o *PoliCap* tem ainda como objetivos viabilizar: a implantação de um primeiro nível de controle de acesso global no Projeto *JaCoWeb Security*; a implementação de políticas locais, através dos objetos *AccessDecision* do

CORBAsec; e a implantação de um controle de acesso dinâmico, determinado em tempo de execução.

Segundo as especificações do modelo CORBAsec, as políticas de segurança tornam-se disponíveis através dos objetos *DomainAccessPolicy* e *RequiredRights*. As interfaces destes objetos permitem o acesso a políticas discricionárias. Aplicações administrativas são responsáveis pela definição desses objetos, enquanto aplicações operacionais utilizam esses objetos para realizar o controle de acesso (OMG, 2000a).

Com as lacunas identificadas nas interfaces de gerência ainda não padronizadas no *CORBAsec*, identificadas no capítulo 3, sentiu-se a necessidade da introdução deste serviço de gerência de objetos de política no esquema de autorização proposto. O serviço desenvolvido - o *PoliCap* - se baseia em documentos *drafts* (propostas iniciais) liberados pela OMG (OMG, 1999a), e certamente não estará muito distante das especificações que deverão em breve ser padronizadas. O *PoliCap* (Figura 3.16) é um serviço que oferece operações tanto para funções *administrativas* quanto *operacionais* sobre objetos de políticas, cumprindo o papel de *gerente de domínio para objetos de política* (*DomainAccessPolicy* e *AuditPolicy*) e de *gerente de direitos para os objetos de direitos requeridos* (*RequiredRights*) do nosso domínio. O esquema de autorização proposto (*JaCoWeb*), inicialmente foi definido como constituído por um único domínio, e o serviço de política atua como *um serviço global de gerenciamento de políticas e de direitos* neste domínio.

Nossa abordagem para a construção do *PoliCap* está baseada no conceito de objetos de serviço da OMG. Como discutido na seção 3.4, uma especificação de um objeto de serviço usualmente consiste de um conjunto de interfaces e de uma descrição do comportamento do serviço, que podem ser utilizados por objetos de aplicação e outros objetos de serviço. A sintaxe usada para especificar as interfaces é a OMG IDL. A semântica que especifica o comportamento do serviço é expressa, em geral, em termos de objetos, operações e tipos de dados padronizados (OMG, 2000a). A figura 4.1 descreve a interface do *PoliCap*. O comportamento associado é descrito nas seções 4.2.1 e 4.2.2.

```
module PoliCap {
#include "SecurityLevel2.idl"

        interface DomainAccessPolicyAdmin:CORBA::DomainManager {

        const CORBA::PolicyType SecClientInvocationAccessDiscretionary  = 100;

        void set_policy ( in        CORBA::PolicyType policyType,
                          in        CORBA::Policy   policy);

        void delete_policy (in      CORBA::PolicyType policyType);

        CORBA::Policy get_local_domain_policy (in       CORBA::PolicyType policyType,
                                               in       Security::SecAttribute      priv_attr,
                                               in       Security::DelegationState   del_state);
        };

        interface RequiredRightsAdmin : SecurityLevel2::RequiredRights {
                SecurityLevel2::RequiredRights get_local_required_rights (
                        in      CORBA::Identifier target_interface_name);
        };
};
```

Figura 4.1 – Interface IDL do *PoliCap*.

4.2.1 Estabelecimento da Política Discricionária usando o PoliCap

Aplicações administrativas interagem com o *PoliCap* para gerenciar as políticas e direitos requeridos. Uma **aplicação administrativa** deve executar alguns procedimentos, usando o *PoliCap*, para criar e definir políticas de segurança.

Primeiramente, deve iniciar o Serviço de Política *PoliCap* e a tabela de privilégios do sistema, definindo a correspondência entre usuários e atributos de privilégio do ambiente CORBA seguro.

O **próprio administrador** do ambiente (executando uma aplicação administrativa), já que ele é quem tem direitos e conhecimento sobre os usuários do sistema, deve:

- definir os objetos *DomainAccessPolicy* e *AuditPolicy* usando as operações da interface *DomainAccessPolicy* e *AuditPolicy*;
- inserir/remover objetos no serviço de política. A operação *set_policy* da *interface DomainAccessPolicyAdmin* associa a política definida e o objeto de política correspondente com o domínio de política. A operação *delete_policy* da *interface DomainAccessPolicyAdmin* remove a política (e o objeto de política correspondente) do domínio.

Na seqüência, o **serviço de política *PoliCap*** deve:

- se **registrar** no serviço de nomes ou outro servidor confiável usado no ambiente[20] para exportar sua IOR para o sistema;
- instanciar os objetos persistentes *DomainAccessPolicy*, *AuditPolicy* e *RequiredRights* existentes.

Os objetos servidores, após se registrarem no serviço de nomes, devem:

- obter a referência do serviço de política;
- especificar a parte do objeto *RequiredRights* que lhes cabe, especificando os direitos requeridos para as operações existentes na sua interface, usando a operação *set_required_rights* da interface *RequiredRightsAdmin* definida no *PoliCap*.

Através da cooperação entre aplicações administrativas, o administrador do ambiente, o serviço de política *PoliCap* e os objetos servidores de aplicação, os objetos *DomainAccessPolicy*, *AuditPolicy* e *RequiredRights*, que compõem a política de autorização discricionária centralizada, são criados e definidos para que possam ser utilizados em tempo de decisão de acesso.

4.2.2 Uso das Políticas Discricionárias Residentes no PoliCap

Aplicações operacionais, ou objetos COSS, interagem com o serviço de política para obter, *em tempo de ligação*, as políticas e os direitos requeridos necessários aos controles em tempo de execução sobre as invocações de um método. A idéia é que, *em tempo de ligação*, o serviço de política (gerente de domínio) forneça os objetos *DomainAccessPolicy, AuditPolicy* e *RequiredRights* que atuam sobre uma invocação (WESTPHALL, FRAGA, *et al.*, 2000).

A operação *get_local_domain_policy* da *interface DomainAccessPolicyAdmin* monta um objeto *DomainAccessPolicy* com os direitos efetivos (*GrantedRights*), presentes na política de autorização na forma de atributo de privilégio e os seus estados de delegação. Na verdade esta operação monta localmente, *em tempo de ligação*, uma versão do *DomainAccessPolicy*, essencial na validação (*em tempo de decisão de acesso*) de diversas operações existentes na interface do objeto servidor referenciado no processo de ligação. Por exemplo, tomando o objeto *DomainAccessPolicy* com as informações da Tabela 4.1 - objeto global definido no domínio -, ao se executar a operação:

```
localDomainAccessPolicy = get_local_domain_policy
        (SecClientInvocationAccessDiscretionary,'role,autôridade,caixa', initiator),
```

[20] Pode ser usado um servidor LDAP, contando com seus serviços de autenticação, para maior segurança.

o retorno desta operação é o objeto *DomainAccessPolicy* - uma versão representada na Tabela 4.2 - que deve atuar localmente no objeto *AccessDecision* de uma invocação.

Atributo de Privilégio	**Estado de Delegação**	**Direitos Fornecidos (*Granted Rights*)**
role:gerente_banco	Initiator	corba: gs--
role:gerente_banco	Delegate	corba: g---
role:caixa	Initiator	corba: g--u

Tabela 4.1 – Objeto *DomainAccessPolicy*.

Atributo de Privilégio	**Estado de Delegação**	**Direitos Fornecidos (*Granted Rights*)**
role:caixa	Initiator	corba: g--u

Tabela 4.2 – Objeto *DomainAccessPolicy* retornado pela operação *get_local_domain_policy*.

O objeto *AuditPolicy* local, por sua vez, é o espelho do objeto global, visto que é dependente do ambiente e, que não existe nenhum critério que simplifique a versão local que é usada durante uma invocação.

A operação *get_local_required_rights* da *interface RequiredRightsAdmin* monta também uma versão local, com os direitos requeridos presentes no objeto *RequiredRights* global (centralizado) do serviço de política do domínio. Esta versão do objeto *RequiredRights*, montada localmente *em tempo de ligação*, contém todas as linhas relativas a uma interface, considerando que um objeto cliente pode executar as diversas operações definidas nesta interface de objeto de aplicação. Por exemplo, tendo-se o objeto *RequiredRights* (global) da Tabela 4.3, ao se executar a operação:

```
localRequiredRights = get_local_required_rights (Renda_fixa),
```

o valor de retorno da operação é o objeto *RequiredRights* (local) representado na Tabela 4.4.

Direitos Requeridos (*Required Rights*)	**Combinador de Direitos**	**Operação**	**Interface**
Corba:g---	All	Ver_Saldo	Renda_Fixa
Corba:gs--	Any	Depositar	Renda_Fixa
Corba:g--u	All	Depositar	Conta_corrente

Tabela 4.3 – Objeto *RequiredRights* para as interfaces Renda_Fixa e Conta_corrente.

Direitos Requeridos (*Required Rights*)	**Combinador de Direitos**	**Operação**	**Interface**
Corba:g	All	Ver_Saldo	Renda_Fixa
Corba:gs	Any	Depositar	Renda_Fixa

Tabela 4.4 – Objeto *RequiredRights* retornado pela operação *get_local_required_rights*.

4.2.2.1 Dinâmica da Autorização usando o PoliCap

Os objetos envolvidos no processo de autorização no esquema de autorização *JaCoWeb* são: *AccessDecision*, *DomainAccessPolicy*, *AuditPolicy*, *RequiredRights*, *SecurityManager*, *PolicyCurrent* e *Current* (seção 3.4). Toda a dinâmica do interceptador de controle de acesso, envolvido no processo de autorização, já foi descrita na seção 3.4.5.1 mas o envolvimento do *PoliCap* neste processo é descrito aqui.

O objeto chave na autorização do CORBAsec usado no esquema de autorização *JaCoWeb* é o objeto de serviço ***AccessDecision*** (seção 3.4.5). O objeto *AccessDecision*, em tempo de decisão de acesso, é o responsável por verificar se uma operação de um objeto destino específico, solicitada por uma aplicação cliente, deve ser permitida ou não. A dinâmica dessa decisão de acesso é ilustrada na Figura 4.2.

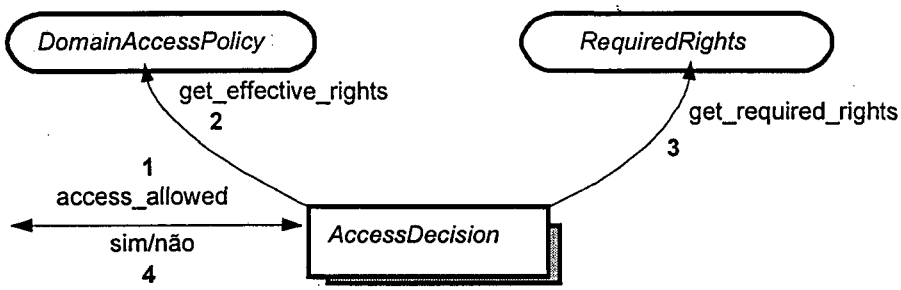


Figura 4.2 – Objeto *AccessDecision*.

As decisões de acesso dependem dos atributos de privilégios do principal, fornecidos pelo objeto ***DomainAccessPolicy*** local, e de atributos de controles do objeto destino, definidos pelo ***RequiredRights*** local.

O interceptador de controle de acesso, através do método ***access_allowed*** do objeto ***AccessDecision*** (passo 1 da Figura 4.2), deve solicitar a verificação de uma invocação de objeto, especificando a operação requerida, as credenciais do principal (que podem ser obtidas pela execução da operação ***get_attributes*** da interface ***Credentials***), o identificador do objeto destino e o nome da interface do objeto destino. No passo 2, o objeto ***AccessDecision***, de posse dessas informações, solicita ao objeto ***DomainAccessPolicy***, através do método ***get_effective_rights***, que informe os direitos concedidos ao principal da invocação, e solicita ao objeto ***RequiredRights***, através do método ***get_required_rights*** (passo 3 da Figura 4.2), os direitos necessários para executar a operação da interface especificada. Já no passo 4, o objeto ***AccessDecision*** compara esses direitos requeridos com os direitos efetivos fornecidos pelos objetos de política, e, caso os direitos requeridos foram fornecidos ao objeto de aplicação, o acesso é permitido. Um exemplo de autorização envolvendo estes objetos foi fornecido na seção 3.4.5.

As políticas de autorização podem ser modificadas pela substituição dos objeto *DomainAccessPolicy* e *AccessDecision*, que definem e garantem o cumprimento das políticas de autorização.

A interação entre os objetos *AccessDecision* e o *Serviço de Política*, em *tempo de binding*, é representado na Figura 4.3. Supõe-se um canal de comunicação seguro durante essa interação. No *JaCoWeb*, esse canal seguro é provido pelo próprio ORB, que realiza conexões seguras usando suporte criptográfico fornecido pelo SSL (seção 4.4.4.4).

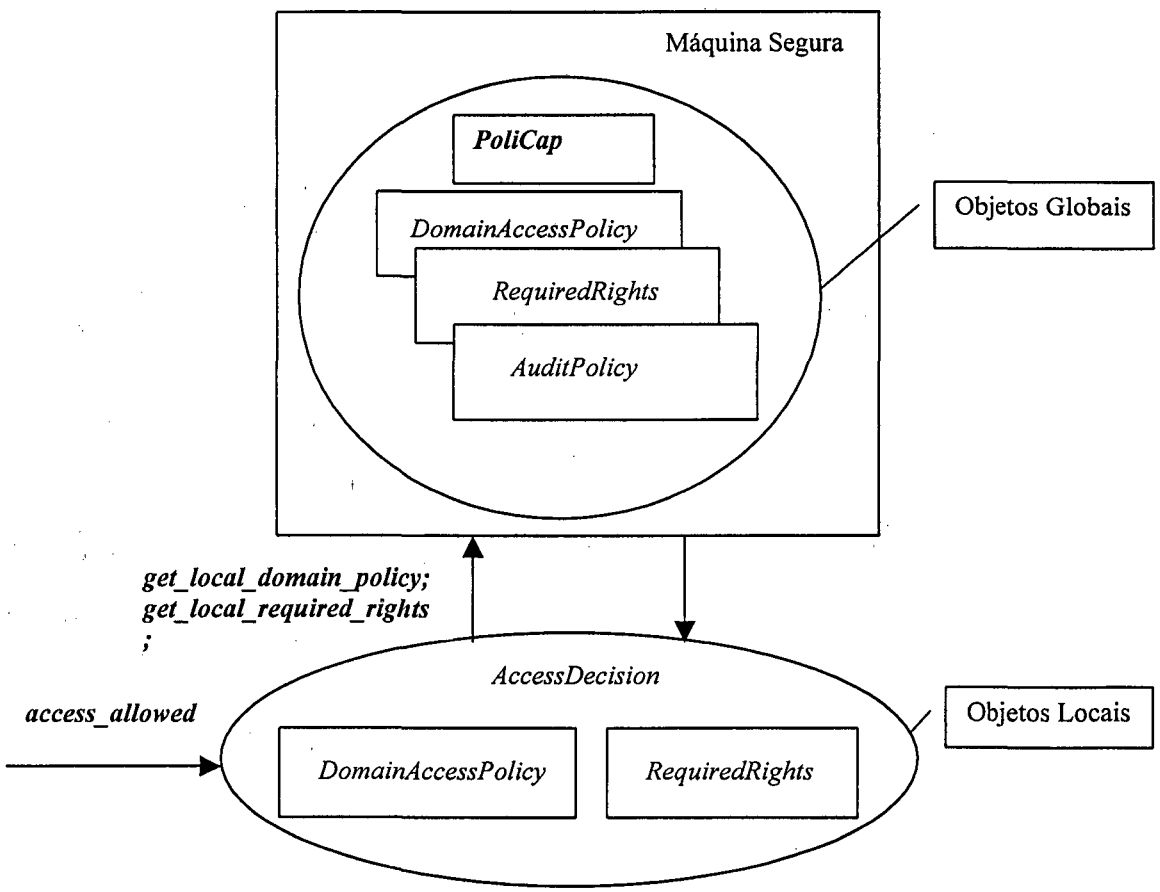


Figura 4.3 – Interação entre o objeto *Access Decision e* o *Serviço de Política.*

A Figura 4.4 resume as ações realizadas em *tempo de ligação* e a Figura 4.5 as ações realizadas em *tempo de decisão de acesso*.

Em *bind time*, o objeto *AccessDecision* deve seguir os seguintes passos (Figura 4.4):
(1) **obter** a referência do serviço de política a partir do serviço de nomes
(2) **criar** um objeto *DomainAccessPolicy* e *AuditPolicy* locais
(3) executar a operação ***get_local_domain_policy*** *(policyType, priv_attr, del_state)*, definida no *PoliCap*, para obter a política de autorização local e a política de auditoria a serem usadas na invocação
(4) armazenar as referências de políticas obtidas no passo anterior no objeto *PolicyCurrent* executando a operação ***set_overrides***
(5) **criar** um objeto *RequiredRights* local
(6) executar a operação ***get_local_required_rights*** *(target_interface_name)*, definida no *PoliCap*, para obter o objeto *RequiredRights* local a ser usado na invocação
(7) **armazenar** a referência do objeto obtido no passo anterior no objeto *SecurityManager*

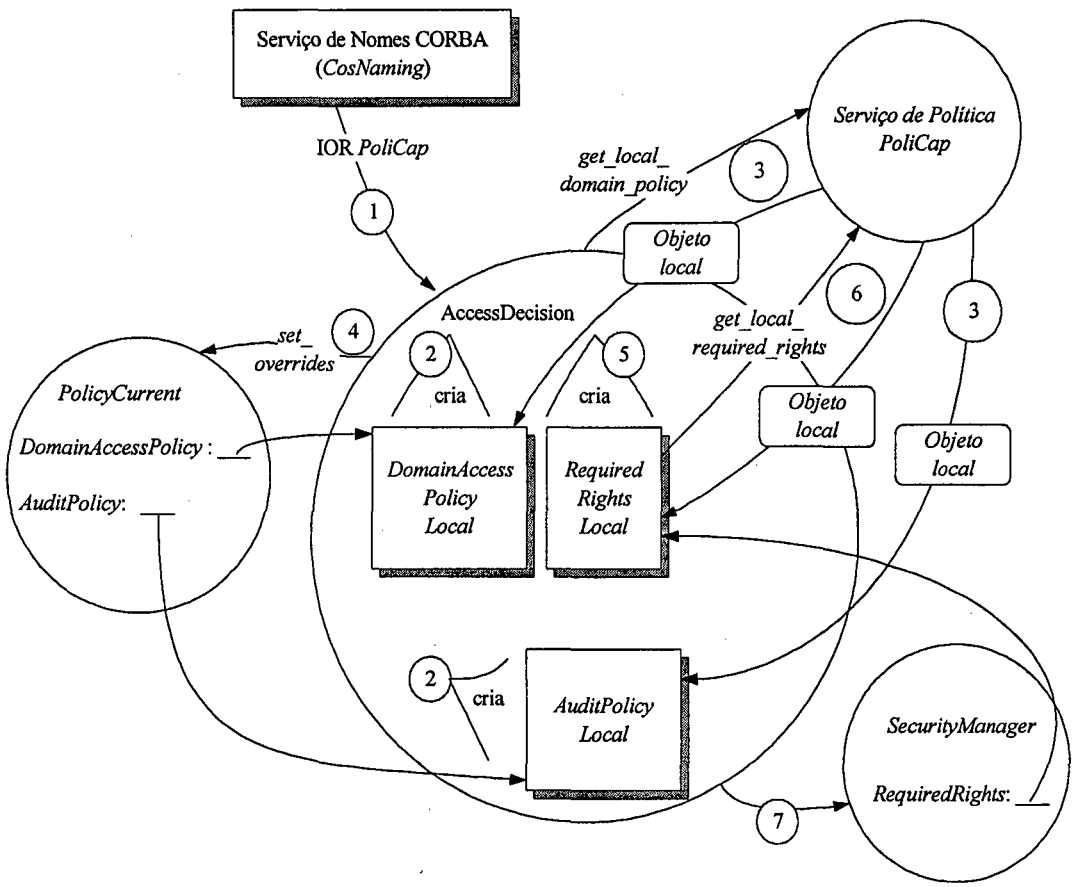


Figura 4.4 – Ações realizadas em *tempo de binding*.

Em *access decision time*, o objeto *AccessDecision* deve seguir os seguintes passos (Figura 4.5):

(1) obter as políticas de autorização (objeto *DomainAccessPolicy*) e de auditoria (objeto *AuditPolicy*) usadas na invocação, executando a operação ***get_overrides*** do objeto *PolicyCurrent*

(2) usar a política obtida no passo anterior para decidir quais direitos o *principal* possui, invocando operações ***get_effective_rights*** no objeto *DomainAccessPolicy* apropriado

(3) o objeto *AccessDecision* deve invocar a operação ***get_required_rights*** do objeto *RequiredRights* (cuja referência está no objeto *SecurityManager*) a fim de encontrar quais direitos são necessários para essa operação. O objeto *AccessDecision* compara esses direitos requeridos (*required_rights*) com os direitos efetivos (*effective rights*) fornecidos pelo objeto de política, e se os direitos conferem o acesso é permitido

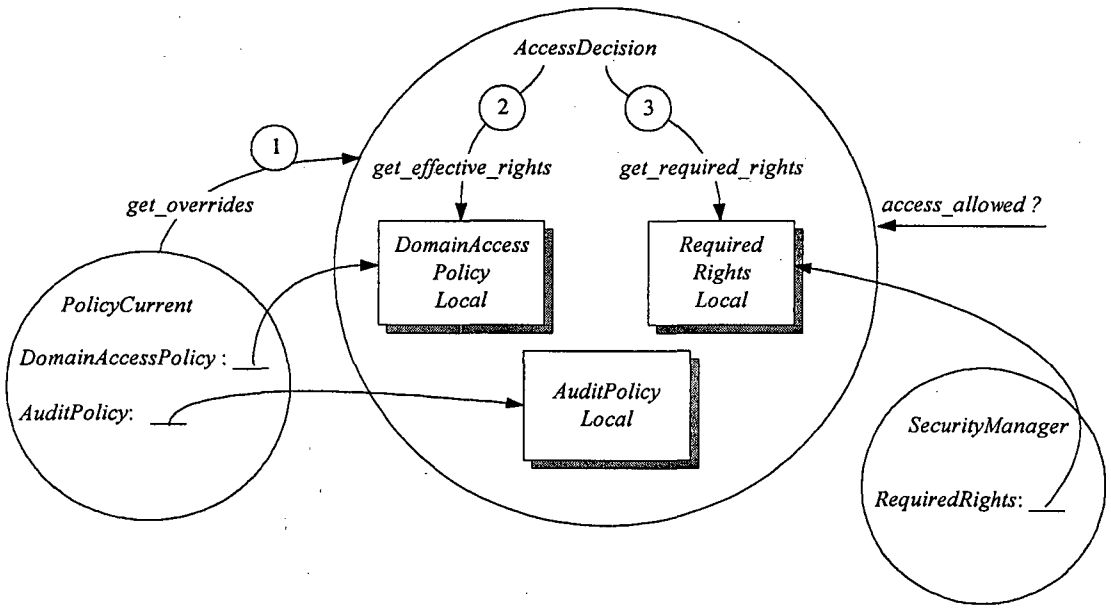


Figura 4.5 – Ações realizadas em *tempo de decisão de acesso*.

4.3 *Capabilities e o Controle de Acesso Local*

No modelo *CORBAsec*, as decisões de controle de acesso podem ser realizadas tanto no lado do cliente como no lado do servidor. O controle de acesso no lado do objeto destino é definido nas especificações do *CORBAsec* como *normal*. Mas, também é salientado que, quando as verificações de controle de acesso são feitas no lado do cliente, as negações de acesso ao objeto servidor determinam uma redução no tráfego de rede. Nota-se também que, em alguns ORBs, considerações de integridade do sistema podem tornar indesejável a confiança no controle de acesso feito unicamente no lado do cliente (OMG, 2000a).

No *JaCoWeb*, o objetivo é realizar o controle de acesso aos objetos distribuídos visíveis via CORBA, e num primeiro nível, as verificações são realizadas pelo objeto *AccessDecision* do

cliente em parceria com o serviço *PoliCap* (nível global de controle de acesso). A execução das operações *get_local_domain_policy* e *get_local_required_rights*, a partir da interceptação de controle de acesso *em tempo de ligação*, monta os objetos *DomainAccessPolicy* e *RequiredRights* locais. Estes objetos fazem parte da estrutura necessária no objeto *AccessDecision* do cliente para gerar *capabilities* (competências) a serem verificadas no objeto *AccessDecision* do servidor. A verificação de nível global só é feita uma vez, durante o *binding* dos objetos cliente e servidor, na primeira requisição. As requisições de operações subseqüentes na interface do servidor serão validadas apenas com o mecanismo de *capabilities*. Os dois níveis de controle de acesso reduzem o tráfego de rede em caso da negação de acesso, e ao mesmo tempo a segurança do sistema não depende unicamente da integridade do cliente.

O mecanismo de competências não é padronizado pelo *CORBAsec*, embora exista a sugestão sobre o uso de *capabilities* em (OMG, 2000a), a partir das abstrações criadas no modelo. Não se pode afirmar que exista uma definição universal *do conteúdo* de uma *capability*. Tradicionalmente, uma *capability* deve conter a referência de um objeto e um conjunto de direitos. A *capability* define os direitos do detentor sobre o objeto em questão. Em (GONG, 1989), uma *capability* clássica é representada como a tripla (*IdObject*, *Rights*, *Random*) contendo o nome do objeto, um conjunto de direitos de acesso e um número aleatório, respectivamente. O número aleatório previne falsificação, sendo normalmente resultado de uma função *one-way f*, aplicada sobre o identificador do objeto e os direitos sobre o mesmo (*Random = f(IdObject, Rights)*)[21]. Quando uma requisição chega no lado do servidor, com a sua respectiva *capability*, a função *one-way f* é executada novamente e o seu resultado é conferido com o número aleatório enviado para detectar possíveis modificações na mensagem da requisição (*tampering*). Este número aleatório que faz o papel de um *nonce*, pode também assegurar também a propriedade de '*freshness*' (ABADI e NEEDHAM, 1996) da requisição do cliente.

Assim como não pode ser possível falsificar uma *capability*, também não deve ser possível reutilizá-la ou repassá-la a terceiros. Para isto, normalmente, além do número aleatório é incluído em uma *capability* um campo identificação do requisitante da operação (o cliente). As *capabilities* são protegidas por mecanismos de cifragem quando transmitidas no suporte de comunicação (GONG, 1989).

[21] Existem vários algoritmos práticos que implementam funções *one-way hash*. Os algoritmos de MD4, MD5 e SHA são exemplo destas funções (STALLINGS, 1998).

Vários trabalhos existentes na literatura (BACON, MOODY, *et al.*, 2000, BROSE, 2000, GONG, 1989, HAGIMONT, HUET, *et al.*, 1997, HAYTON, BACON, *et al.*, 1998) utilizam *capabilities* em seus sistemas de segurança. Essas *capabilities* representam, em um sentido tradicional, os direitos pertencentes a um *principal* ou requisição que possibilita o acesso a um objeto, e são estabelecidos ou de forma estática, em (BROSE, 2000, HAGIMONT, HUET, *et al.*, 1997) ou de forma dinâmica, em (BACON, MOODY, *et al.*, 2000, GONG, 1989, HAYTON, BACON, *et al.*, 1998). A vantagem do uso de *capabilities* é a sua flexibilidade em permitir a evolução dinâmica dos direitos de acesso no sistema (HAGIMONT, HUET, *et al.*, 1997).

No esquema *JaCoWeb*, as *capabilities* são criadas dinamicamente a cada requisição do cliente para operações no servidor. A classe *Request* (estrutura do pedido) definida nas especificações CORBA é usada na composição de uma *capability* no nosso esquema. Uma *capability* no *JaCoWeb* contém nos seus campos: a *IOR* ("*Interoperable Object reference*") do objeto servidor, representando a identidade da entidade sobre a qual a *capability* fornece os direitos de acesso; *método solicitado*, representando o direito; *identificador do emissor/principal*, representando a identidade da entidade solicitante da operação; e um *nonce* que assegura a propriedade '*freshness*' do pedido. Em um *Request* de uma invocação usual no CORBA, o *IOR* do objeto servidor e a identificação do método solicitado já estão presentes. Os outros dois campos da *capability*, identificador do emissor do pedido e um campo de *nonce*, devem ser inseridos no *Request* durante a interceptação de alto nível, no objeto *AccessDecision* do lado do cliente. O valor de *nonce* é calculado da seguinte forma:

nonce = *f*(identificador do emissor, método solicitado, IOR do servidor, Random);

onde *f* corresponde à função *hash one-way* SHA, presente no pacote *java.security* (SUN, 1999). O valor *Random* é um número aleatório calculado fazendo uso da classe *java.security.SecureRandom* da API Criptográfica do Java JDK 1.2 (SUN, 1999). Este valor aleatório é uma redundância que garante a propriedade *freshness* da mensagem do *Request*, garantindo contra os ataques por mensagens antigas (GONG, 1989). O valor inicial de *Random* é obtido durante o estabelecimento da associação segura, e é conhecido entre cliente e servidor, atuando também como uma *chave de sessão* entre ambos. Os valores de *Random* posteriores funcionam como um contador de mensagens. Todos os pedidos de operações do cliente sobre a interface do servidor, terão as mensagens correspondentes enviadas pela associação segura, seguindo a numeração de seqüência *Random*, *Random+1*, *Random+2*,... .

O identificador do emissor (ou solicitante da operação) é conseguido a partir da credencial do cliente. Depois das inclusões para gerar a *capability*, o *Request* fica representado como na estrutura da Figura 4.6.

Request novo com a *capability* (IOR, método, emissor, *nonce*)				
Request CORBA			Campos adicionados	
IOR servidor	Método solicitado	...	Identificador Emissor	*Nonce*

Figura 4.6 – Estrutura do *Request* com campos adicionais para implementar a *capability*.

Em *tempo de decisão de acesso*, o interceptador de controle de acesso *no lado do servidor* faz a verificação da *capability*. Dessa forma pode-se construir *capabilities* para cada uma das invocações de um cliente no *CORBAsec*. Essas *capabilities* são utilizadas para formar o segundo nível de controle de acesso no esquema de autorização *JaCoWeb*.

4.3.1 Segurança do Ambiente Java no Controle de Acesso Local

Os núcleos de segurança e *Trusted Computing Bases* validam os acessos locais dos *applets* de aplicação. Para implementar essa base confiável que valida os acessos remotos, usamos os objetos de decisão de acesso *AccessDecision* e os interceptadores (serviços ORB) do CORBAsec, responsáveis pela geração e verificação das *capabilities*. Nos acessos locais são usados o modelo de segurança Java com seus procedimentos de controle de acesso.

O modelo de segurança Java, na sua versão 1.2, possui arquivos de política que identificam quais operações podem ser realizadas pelos códigos carregados (*applets*) (Figura 4.7). A classe *java.security.Policy* estabelece uma política de segurança para o ambiente de execução Java, representada por um objeto *Policy*. Para verificar as permissões que podem ser criadas, o objeto *Policy* usa um arquivo de política corrente que contém essencialmente uma lista de entradas. Cada entrada possui campos *keystore* e *grant*. O *keystore* indica a localização da entidade que pode validar as chaves públicas do *applet*, passadas com o objeto *CodeSource*. O campo *grant* define para o *applet* com a URL definida no *CodeSource* o conjunto de permissões que podem ser criados para o mesmo. Como exemplo, na Figura 4.7, é representado um arquivo de política corrente.

```
keystore http://foo.bar.com/blah/some.policy
grant codeBase "http://java.sun.com", signedBy "Li" {
                permission java.io.FilePermission "/tmp/*", "read";
                permission java.io.SocketPermission "*", "connect";
}
```

Figura 4.7 – Arquivo de Política do Modelo de Segurança Java JDK 1.2.

Além disso, conta com o gerente de segurança (*Security Manager Class*) que realiza o controle de acesso baseado na política concretizada através dos domínios de proteção (*Protection Domains*). A política de segurança do sistema, definida pelo usuário ou pelo administrador do

sistema, especifica quais domínios de proteção devem ser criados e quais permissões devem ser fornecidas para esses domínios internos, na máquina virtual Java. O ambiente de execução Java mantém um mapeamento do código (classes e instâncias) para seus domínios de proteção e permissões correspondentes, conforme ilustrado na Figura 4.8.

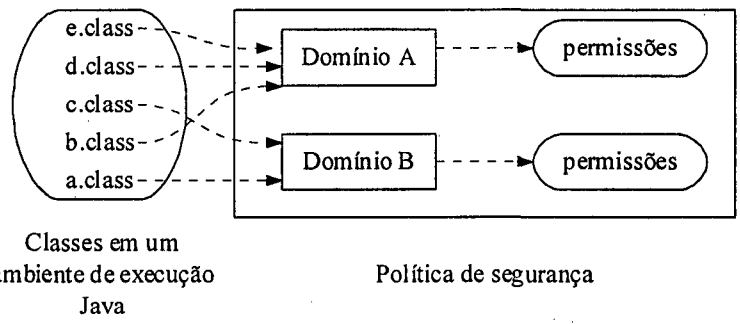


Figura 4.8 – Mapeamento da política de segurança.

Infelizmente, a implementação padrão do modelo de segurança Java JDK 1.2 não trata das necessidades administrativas existentes em sistemas distribuídos (NIKANDER e PARTANEN, 1999). O arquivo de política é usado para descrever como as permissões são fornecidas para cada classe, baseado na assinatura e localização da classe carregada. Em termos práticos, isso significa que este arquivo deve contar com todas as combinações de permissões possíveis e com as chaves e configurações de segurança necessárias, e deve estar presente em cada máquina usada em uma configuração de execução do ambiente Java distribuído. Se houver necessidade de modificação desse arquivo de política, todos os arquivos locais devem ser atualizados em cada uma das máquinas.

Dessa forma, neste trabalho, optamos por uma solução apresentada em (NIKANDER e PARTANEN, 1999), onde o arquivo de política é definido através de uma URL e assim, o arquivo pode ser carregado, usando o suporte criptográfico do próprio browser (SSL), a partir de um servidor Web durante a execução do ambiente Java.

4.4 Implementação do Esquema de Autorização Discricionário

4.4.1 Aplicação Desenvolvida para Validar o Protótipo

O comércio eletrônico ainda está iniciando a sua expansão no Brasil, representando 15 % das transações eletrônicas realizadas. A principal aplicação, com 27 % de uso, é o *Home/Internet Banking* (MÓDULO, 1999). De acordo com os últimos congressos sobre automação bancária, o uso de serviços de *Internet Banking* vem surgindo como solução para a crescente necessidade de serviços mais eficazes, mas somente cerca de 50 % dos bancos brasileiros oferecem este serviço.

Por ser uma aplicação crítica amplamente utilizada em ambientes de larga escala, escolheu-se um sistema de *Internet Banking* para avaliar a potencialidade do esquema de autorização discricionário, proposto neste capítulo.

Na implementação do Projeto *JaCoWeb Security*, ou seja, dos *applets* de aplicação e servidores de aplicação que usam o pacote *JaCoWebSecurity*, foi utilizado um sistema de *Internet Banking* fictício, composto por um objeto servidor bancário CORBA, um *applet* cliente Java e uma aplicação *stand-alone* cliente responsável por algumas funcionalidades administrativas do sistema bancário (Figura 4.9).

A partir de qualquer ponto da rede Internet, com o auxílio de um navegador, um cliente do banco pode se comunicar com o seu servidor bancário. Um cliente deve fornecer o endereço da página *Web* do banco, para que a carga automática do *applet* seja realizada na estação cliente. Quando uma operação é solicitada, o *applet* cliente se comunica com um serviço de nomes CORBA, para obter a referência do objeto servidor bancário. De posse dessa referência, e se for um usuário autorizado, o cliente estabelece uma associação segura com o objeto CORBA servidor. Nesta implementação, o servidor Web que armazena o código do *applet* cliente, o objeto CORBA (servidor bancário) e o serviço de nomes CORBA não necessitam estar na mesma máquina.

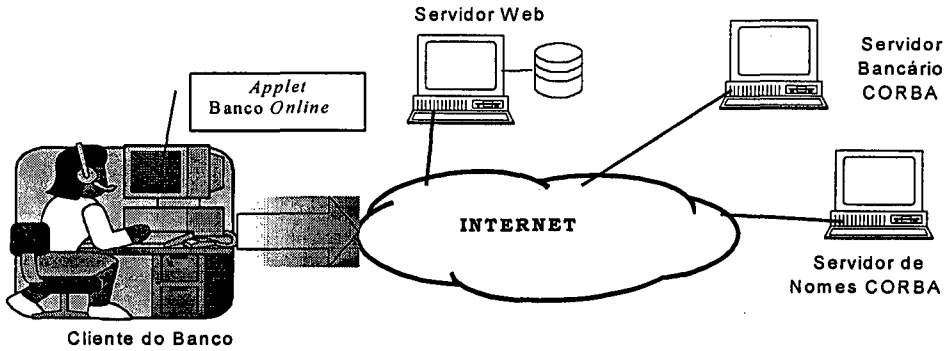


Figura 4.9 – Estrutura do Protótipo Implementado (WANGHAM, 2000).

A funcionalidade provida pela aplicação bancária é representada com o uso do *usecase* UML (ERIKSSON e PENKER, 1998) na Figura 4.10. Um servidor bancário serve clientes do banco e gerentes do banco. Tanto clientes quanto gerentes gostariam de armazenar seus dados de forma segura de modo que outros usuários e não-usuários em geral não tivessem acesso a esses dados. Além disso, o próprio banco armazena informações privadas como dados dos seus investimentos, que não devem ser vistos por usuários e não-usuários. O banco recebe pedidos periódicos de cada cliente para verificar o extrato, verificar o saldo e fazer transferência de dinheiro entre contas.

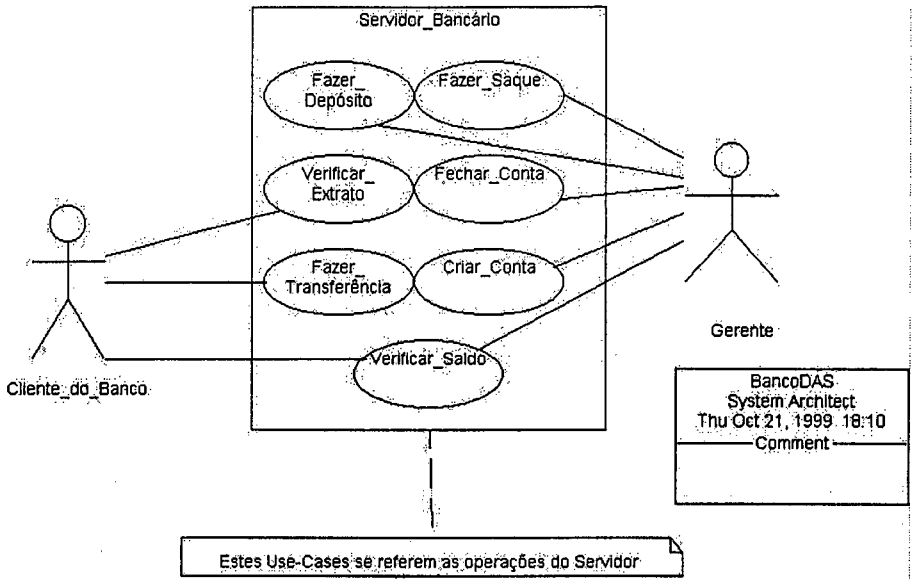


Figura 4.10 – Diagrama *Use-Case* da Aplicação Bancária.

4.4.2 Modelo do Protótipo Implementado

Um protótipo - incluindo o mecanismo de *capabilities* e uma versão simplificada do serviço de políticas *PoliCap* - foi desenvolvido em nosso laboratório (Figura 4.11). Uma aplicação exemplo constituída de um sistema bancário composto por um objeto servidor CORBA e um *applet* cliente Java foi construída para testar as implementações deste protótipo (ver seção 4.4.4). O objeto servidor CORBA foi desenvolvido com a ferramenta JacORB 1.0 (BROSE, 1997), um ORB Java livre (*free*), e o *applet* cliente e a aplicação *stand-alone* foram implementados com a ferramenta JDK 1.2.1. O *browser* Netscape Communicator 4.5 também foi usado como ambiente para a interação entre o cliente e o servidor. Para o estabelecimento da associação segura foi empregado, como tecnologia de segurança subjacente, o protocolo SSL. O objetivo dessa implementação foi concretizar uma política de controle de acesso discricionária baseada nas estruturas definidas no *CORBAsec* (WESTPHALL, FRAGA, *et al.*, 2000).

O modelo do protótipo do esquema de autorização *JaCoWeb* discricionário implementado apresenta os objetos do modelo CORBAsec que foram implementados, a tecnologia de segurança subjacente que foi utilizada e outras características. Este protótipo, cuja estrutura está representada na Figura 4.11, é uma simplificação da arquitetura do esquema de autorização *JaCoWeb* completo apresentado na Figura 3.16.

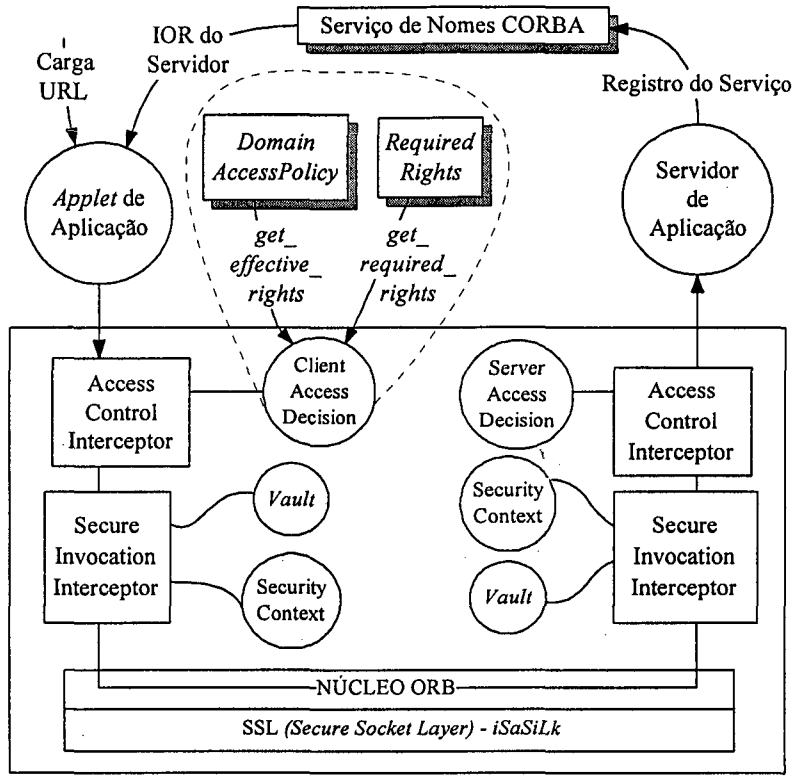


Figura 4.11 – Modelo do Protótipo Implementado.

Essa versão do *PoliCap* preocupou-se em amadurecer, primeiramente, todo o aspecto dinâmico dos objetos de serviço do CORBAsec, o que não é uma tarefa trivial. Isso porque a especificação utilizada é extremamente ampla (OMG, 2000a), e o dinamismo de uma aplicação que usa o CORBAsec prevê o entendimento de um grande número de objetos, o que não é descrito na literatura. Portanto, foi feito um experimento no qual os objetos *DomainAccessPolicy* e *RequiredRights* locais são estabelecidos em tempo de ligação entre cliente e servidor de forma estática. A inserção, atualização e remoção de direitos nesses dois objetos (*DomainAccessPolicy* e *RequiredRights*) são executadas pelos próprios objetos servidores da aplicação, logo depois de se registrarem no serviço CORBA de nomes (*CosNaming*).

Dentre os objetos implementados nesse protótipo (além do *applet* e servidor da aplicação) estão os objetos *ClientAccessDecision*, *ServerAccessDecision*, *Vault*, *SecurityContext*, *DomainAccessPolicy*, *RequiredRights*, *SecurityManager* e *PolicyCurrent* do modelo CORBA de segurança. Esses objetos usam outros serviços CORBA (COSS) como o serviço de nomes. O protótipo também está limitado a um único domínio de nomes.

As entidades que estão sujeitas aos controles de segurança no nosso sistema são os principais (com seus atributos de privilégio) e os objetos servidores da aplicação (com seus atributos de controle). O experimento implementa políticas discricionárias, sendo que o primeiro nível de verificação de controle de acesso é feita no lado do cliente e o segundo nível é feito no lado do servidor.

A assinatura digital do modelo de segurança do Java foi utilizada para permitir aos clientes acesso ao disco local ou conexões com máquinas diferentes a partir das quais foram carregados, livrando os *applets* das restrições impostas pelo modelo *sandbox* do Java, permitindo comunicação com quaisquer objetos sem limite de localização. Dessa forma o servidor Web e o servidor da aplicação não necessitam permanecer na mesma máquina.

O protocolo SSL (*Secure Socket Layer*) foi empregado como tecnologia de segurança subjacente. Dentre as várias tecnologias de segurança em sistemas distribuídos, o SSL, desenvolvido pela Netscape, se destaca por ser um protocolo criptográfico amplamente utilizado em aplicações na Internet. Este protocolo é utilizado neste trabalho por estar inserido pela OMG, em sua última revisão do serviço de segurança, como um dos protocolos a serem utilizados por aplicações que implementam a segurança no CORBA, mesmo que a integração de protocolos 'plugáveis' (*pluggable*), como o SSL, não estejam totalmente padronizados pela OMG (ALIREZA, LANG, *et al.*, 2000, OMG, 2000b, 2000c), como já foi discutido no capítulo 3.

O nível de funcionalidade de segurança que se espera atingir com o esquema de autorização implementado é o nível 2, cujas interfaces IDL encontram-se no módulo *SecurityLevel2* do CORBAsec. O gerenciamento das políticas discricionárias de segurança é abordado neste trabalho com base no módulo *SecurityAdmin* da especificação CORBAsec. Entretanto, admite-se que, como as aplicações não estarão cientes da segurança que lhes será atribuída, estas não poderão introduzir suas regras para impor suas próprias políticas. Ou seja, as interfaces definidas para a aplicação não serão tratadas no protótipo aqui proposto. A implementação do protótipo está detalhada na seção 4.4.4.

4.4.3 Ferramentas de Programação

As ferramentas de programação utilizadas na implementação (WANGHAM, 2000), foram o JacORB 1.0 (BROSE, 1997), o JDK 1.2.1 (OAKS, 1999, PISTOLA, RELLER, *et al.*, 1999, SUN, 1998) e o pacote *iSaSiLk* que implementa o protocolo SSL v3 (GRAZ, 1999a). O JDK 1.2.1 já foi discutido no capítulo 3 no contexto do modelo de segurança Java.

4.4.3.1 JacORB

Toda a implementação do protótipo teve como plataforma de desenvolvimento o JacORB beta 13 (BROSE, 1997), protótipo desenvolvido na Universidade de Berlim disponível em http://www.inf.fu-berlin.de/~brose/jacorb/. Esse *middleware* corresponde a um conjunto de suportes e ferramentas de desenvolvimento que permitem construir e integrar aplicações orientadas a objetos em sistemas abertos. O JacORB é completamente desenvolvido em Java, trazendo as vantagens desta linguagem, seguindo as especificações da OMG. Todos os componentes do

JacORB se comunicam usando o protocolo IIOP, também padronizado pela OMG. Além disso, o JacORB, dentre os vários serviços que possui, apresenta como destaque o suporte *multithread*, o POA (*Portable Object Adapter*), *gateway* para comunicação dos *applets* e interceptadores.

Quando um cliente faz uma invocação de método em um objeto servidor remoto, há todo um conjunto de procedimentos no tratamento da requisição para que esta possa ser transmitida no canal de comunicação até chegar ao servidor. Uma requisição do cliente é transformada em um objeto *Request*, o qual contém um conjunto de atributos que define o objeto destino (o servidor), a operação requisitada (o método), os argumentos da invocação, a resposta, entre outros; e um conjunto de métodos que permitem manipular esses atributos. Uma vez criado pelo ORB, a partir de uma requisição do cliente, o objeto *Request* passa pelo processo de conversão (*marshalling*) para *byte stream* para que possa ser enviado pela camada de transporte.

Na Figura 4.12, é apresentado o modelo de implementação do ORB/CORBA do projeto *JacORB*. Em detalhe, é apresentado o fluxo de uma invocação dentro do ORB, a partir do ponto em que os dados da requisição (em *byte stream*) alcançam o *socket* no lado servidor. O módulo *BasicAdapter*, através do objeto *Connection*, trata das funções de baixo nível que manipulam as conexões TCP/IP entre clientes e servidor (os objetos *ServerSocket*). É importante ressaltar que é criada apenas uma conexão para cada par cliente/servidor, não importando quantos objetos no servidor estão sendo utilizados pelo cliente. O objeto *RequestReceptor* faz a decodificação (*unmarshalling*) do conjunto de *byte stream* para remontar o objeto *Request* no lado servidor. Então, o *Request* é transformado em *ServerRequest* para ser mandado para a fila de requisições (*RequestQueue*), onde é ordenado em seqüência para chegar ao POA (*Portable Object Adaptor*).

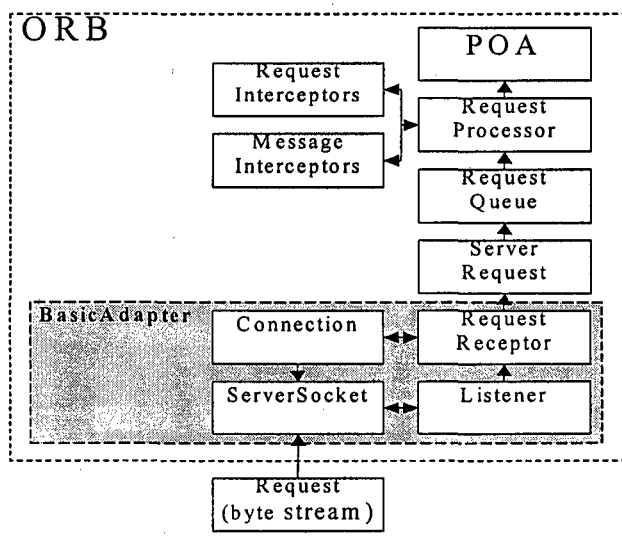


Figura 4.12 – Fluxo de dados no JacORB.

Por definição, o POA é o adaptador de objetos portável do modelo CORBA que é responsável, em conjunto com o ORB, por determinar qual *servant* deve ser invocado e executar a

invocação, ativando a implementação correta (OMG, 1999b). Um *servant* é uma entidade de uma linguagem de programação particular (por exemplo, um objeto em Java ou uma coleção de funções em C). Um objeto CORBA é uma representação remota de um objeto *servant* diante do ORB – e permite, deste modo, que o *servant* possa ser identificado, localizado e endereçado por uma referência de objeto (IOR). As principais funções do POA são geração e interpretação de referência de objeto, invocação de métodos, ativação, desativação de objeto e implementações, e mapeamento de referência de objeto à implementação.

4.4.3.2 O Protocolo SSL e o pacote iSaSiLk

O SSL (FREIER, KARLTON, *et al.*, 1996) é um protocolo de propósito geral adequado para proteger conexões em sistemas distribuídos, prover autenticação, confidencialidade e integridade para comunicações através de conexões TCP/IP. Este protocolo está localizado entre o protocolo da camada de transporte (acima do TCP) e o protocolo da camada de aplicação, e é composto por duas camadas: o protocolo ***SSL Record***, que provê conexões seguras garantindo confidencialidade e integridade das mensagens transportadas, através de criptografia simétrica e de mensagens de integridade (MAC – *message authentication code*); e o protocolo ***SSL Handshake***, que permite a autenticação de cliente e servidor, negocia algoritmos criptográficos e gera a chave simétrica usada pela camada *SSL Record* (ver Figura 4.13).

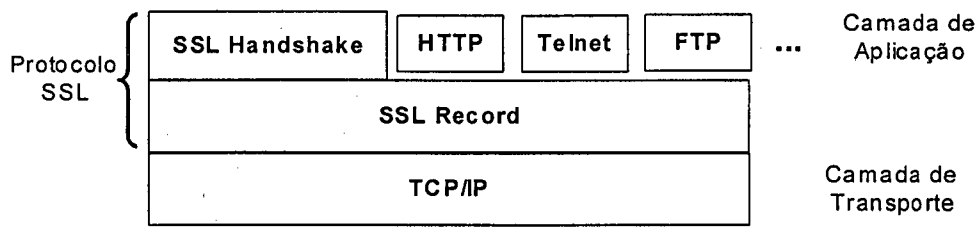


Figura 4.13 – Arquitetura SSL.

O protocolo **SSL *Record*** é usado para transferir dados de aplicação e de controle SSL entre cliente e servidor. Estes dados estão possivelmente fragmentados em pequenas unidades, e podem ser comprimidos, assinados e cifrados antes de serem transmitidos pelo protocolo de transporte confiável (TCP).

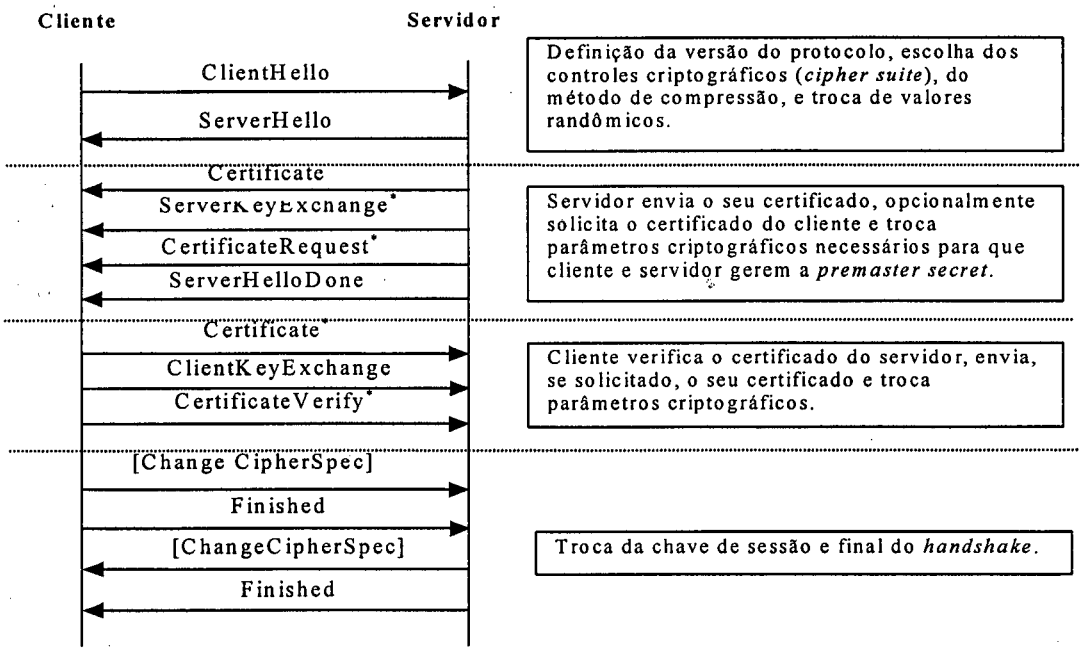
O SSL utiliza, para autenticação de cliente e servidor, certificados digitais e transferências com assinaturas digitais. A sessão SSL é estabelecida quando a negociação inicial (*handshake*) entre cliente e servidor é realizada. O fluxo de mensagens durante o *handshake* está representado na Figura 4.14.

A autenticação no SSL usa os certificados X.509 (ITU-TX509, 1993) para transferir informações sobre a chave pública de uma aplicação. O SSL v3 usa os certificados X.509 v3 para assegurar chaves públicas RSA, e um certificado X.509 modificado para assegurar chaves públicas,

usadas pelo protocolo de distribuição de chaves do Departamento de Defesa dos EUA, *Fortezza/DMS*.

O X.509 *certificate* possui autoridades certificadoras localizadas no mundo todo. O X.509 é um padrão internacional que tem aplicabilidade em diversos campos, com forte aceitação internacional.

No protótipo, foram utilizadas as funções criptográficas IAIK-JCE e o pacote iSaSiLk v.5.2 que implementa a versão atual do protocolo SSL (SSL v3) em Java (disponível em (GRAZ, 1999a, 1999b)). Este pacote foi desenvolvido no *Institute for Applied Information Processing and Communications* da Universidade Tecnológica de Graz, na Áustria, e pode ser obtido sem custo, para fins não comerciais, em http://jcewww.iaik.at/products/isasilk/index.htm. Escolheu-se o pacote iSaSiLk (GRAZ, 1999a) por apresentar todas as funcionalidades necessárias para o protótipo proposto. Toda a negociação e uso do SSL, no protótipo, é detalhada na seção 4.4.4.4.



* Indica que a mensagem é opcional, ou depende da situação para ser enviada.

Figura 4.14 – Fluxo de mensagens do *handshake*.

4.4.4 Implementação do Esquema de Autorização JaCoWeb Discricionário

A implementação do Esquema de Autorização *JaCoWeb* Discricionário é formada por vários aspectos relacionados:

- com a inicialização da estrutura de segurança, ou seja, a inicialização dos objetos do modelo CORBAsec (seção 4.4.4.1);
- com o estabelecimento de credenciais e certificados que são usados em uma associação segura (seção 4.4.4.2);
- com a autorização do pedido de acesso, de acordo com a política discricionária, executada em ambos lados do cliente e do servidor (seção 4.4.4.3);
- com a integração da tecnologia de segurança ao ORB (seção 4.4.4.4).

O trabalho de implementação foi iniciado em âmbito de uma dissertação de mestrado (WANGHAM, 2000, WANGHAM, LUNG, *et al.*, 2000), com o objetivo de concretizar as propostas feitas em (WESTPHALL e FRAGA, 1999a, 1999b).

4.4.4.1 Inicialização do ORB seguro

Para a configuração do ORB seguro no *SecurityLevel2* é necessário definir um conjunto de propriedades, que poderá ser usado para configurar os protocolos de segurança, os mecanismos e outras características da inicialização do canal seguro. Caso este conjunto de propriedades não seja definido, devem-se usar configurações previamente definidas como padrão.

Após a inicialização do ORB e do POA (somente para o servidor), deve ocorrer a inicialização da estrutura de segurança do modelo proposto, definida pela obtenção do objeto de sessão *Current*, que fornece e armazena todas as informações do contexto de execução da associação segura estabelecida. Como a versão atual do JacORB ainda não comporta os objetos de sessão *SecurityManager* e *PolicyCurrent*, definidos na especificação mais atualizada do CORBAsec e no modelo da Figura 3.16, a funcionalidade destes dois objetos é executada pelo objeto *Current* no JacORB. Para finalizar essa inicialização, a classe responsável pela inicialização do ORB seguro fornece uma coleção de métodos, definidos estaticamente, que são necessários para a criação de objetos que representam as configurações da associação segura a ser estabelecida.

Em (ADIRON, 2000) tem-se a definição de algumas propriedades que podem ser usadas pelo programador da aplicação para a inicialização de um ORB seguro. Com base nessas propriedades definiu-se, para o projeto *JaCoWeb Security*, a propriedade jacoweb.seciop usada para especificar onde o protocolo SECIOP deve ser habilitado (cliente e/ou servidor) e as propriedades jacoweb.seciop.host e jacoweb.seciop.port que especificam um sítio (*host*) e uma

porta de comunicação para conexões SECIOP, respectivamente. Para definir o uso do protocolo SSL, tem-se a propriedade jacoweb.ssliop. E, para usar comunicações CORBA padrão (não seguras), definiu-se a propriedade jacoweb.iiop que especifica onde o protocolo IIOP deve ser habilitado.

4.4.4.2 Credenciais e Certificados

O objeto *PrincipalAuthenticator*, responsável pela autenticação e criação do objeto *Credential*, não foi implementado nesse protótipo. As credenciais do protótipo são criadas estaticamente (a partir dos certificados), e possuem atributos de segurança que identificam os direitos do cliente no sistema (atributo de privilégios).

As credenciais usadas neste experimento são formadas pelo atributo de privilégio ***role*** da família CORBA. O valor deste atributo (*value*) e a autoridade que o valida (*defining_authority*) são obtidos a partir dos certificados[22] do principal. Os campos *value* e *defining_authority* são obtidos a partir do campo *subjectDN* e *issuerDN* do certificado, respectivamente. Conforme sugerido na especificação do CORBAsec, estes campos são codificados através da classe *Opaque*, que implementa um simples algoritmo para cifragem dos dados.

Os certificados usados nesta implementação foram criados utilizando como ferramenta o pacote JCE-IAIK. Todos os clientes do sistema bancário, usuários do *internet banking* (*clienteWeb*) ou gerentes, devem possuir certificados, tendo como emissor o *Banco*. Para o campo *subjectDN* é definido um *role*, que pode ser *clienteWeb* ou *gerente*, representando o atributo de privilégio do cliente. Para o *Banco* foi criado um certificado auto-assinado[23].

4.4.4.3 Controle de Acesso Discricionário e Capabilities

O modelo de implementação é representado na Figura 4.11. A implementação usa como mecanismos de desvio os *interceptors* presentes no JacORB. No protótipo, o interceptador de controle de acesso na máquina do cliente tem como função aplicar a política de autorização discricionária definida nos objetos *DomainAccessPolicy* e *RequiredRights*. Este interceptador invoca o objeto *ClientAccessDecision* que é responsável pela validação dos pedidos de acesso aos métodos do objeto servidor, interagindo com os objetos *DomainAccessPolicy* e *RequiredRights*. O método *access_allowed* do objeto *ClientAccessDecision* (Figura 4.15) obtém os direitos requeridos invocando o método *get_required_rights* do objeto *RequiredRights*, e obtém os direitos fornecidos pela política *DomainAccessPolicy* invocando o método *get_effective_rights*. Compara os direitos

[22] Um certificado é formado pelas seguintes informações: sujeito do certificado (*subjectDN*); o emissor do certificado (*issuerDN*); período de validade; informações administrativas (versão e número de série) e informações extras.
[23] Em um certificado auto-assinado o emissor e o sujeito são os mesmos.

requeridos e os direitos fornecidos ao atributo de privilégio para decidir se o método a ser invocado pode ou não pode ser executado.

```
/**
 * Decisao de acesso do lado do cliente.
 * Creation date: (31-07-2000 %T)
 */
public class ClientAccessDecision implements Org.omg.SecurityLevel2.AccessDecision {
        protected Org.omg.SecurityLevel2.RequiredRights rights;
        private Org.omg.Security.Right[] granted;
        private AccessPolicy access = new AccessPolicy();
/**
 * access_allowed method comment.
 */
public boolean access_allowed(Org.omg.SecurityLevel2.Credentials[] cred_list, Org.omg.CORBA.Object target,
String operation_name, String target_interface_name) {

boolean b = false;
// Obtém os direitos requeridos – invoca get_required_rights

Org.omg.Security.RightsListHolder rightslist = new Org.omg.Security.RightsListHolder();
Org.omg.Security.RightsCombinatorHolder combinator = new Org.omg.Security.RightsCombinatorHolder();

// tratar o erro ao obter os direitos (right vazio) da operacao
rights.get_required_rights((Org.omg.CORBA.Object) null, operation_name, "sistemaBancario.idl", rightslist,
combinator);

// extensiblefamily do projeto JaCoWeb - ver classe SSLCredentials
Org.omg.Security.ExtensibleFamily extFamily = new Org.omg.Security.ExtensibleFamily((short) 0, (short) 1);
Org.omg.Security.AttributeType[] attributes = new Org.omg.Security.AttributeType[1];
attributes[0] =  new Org.omg.Security.AttributeType(extFamily, 5);

// Obtém os direitos fornecidos pelo objeto DomainAccessPolicy – invoca get_effective_rights
granted = access.get_effective_rights(cred_list[0].get_attributes(attributes), extFamily);
for (int i = 0; i < access.getCount(); i++) { // Compara direitos para permitir ou negar o acesso
                if (granted[i] != null) {
                        if ((granted[i].right).equals(rightslist.value[0].right)) {
                                b = true;
                                break;
                        }
                }
        }
        return b;
}
```

Figura 4.15 – Método *access_allowed* do objeto *ClientAccessDecision.*

A partir dessa verificação de alto nível, o objeto *ClientAccessDecision* em cooperação com o interceptador de controle de acesso (que tem acesso à estrutura do *Request* CORBA), gera *capabilities* (Figura 4.16) que são verificadas no lado do servidor pelo objeto *ServerAccessDecision*, concretizando o segundo nível de controle de acesso.

```
/**
 * Insere a capability no request do cliente.
 */
public void pre_invoke(Org.omg.CORBA.Request request) {

jacorb.orb.dii.Request r = (jacorb.orb.dii.Request) request;
String method = request.operation();
boolean b = false;
Current cur = (Current) jacorb.orb.Environment.getSecurityConfig().client_current;
cad = cur.access_decision();
SSLCredentials[] creds = { (SSLCredentials) ((cur.own_credentials())[0]) };
try {
        b = cad.access_allowed(creds, (Org.omg.CORBA.Object) null, method, "sistemaBancario.idl");
        if (b) { // se o acesso for permitido, gera a capability
                emissor_id = getEmissor(creds);
                iaik.java.security.MessageDigest sha = iaik.java.security.MessageDigest.getInstance("SHA");
                byte[] random_bytes = new byte[4];      // deveria ser obtido do servidor de politicas
                random_bytes[0] = (byte) (random & 0xff);
                random_bytes[1] = (byte) ((random >>> 8) & 0xff);
                random_bytes[2] = (byte) ((random >>> 16) & 0xff);
                random_bytes[3] = (byte) ((random >>> 24) & 0xff);
                random++;

                // gera capability
                sha.update(emissor_id);
                sha.update(method.getBytes());
                sha.update(request.target().toString().getBytes());
                sha.update(random_bytes);
                nonce = sha.digest();
                r.args_changed = true;
        } else {
                System.out.println("Access denied for operation: " + method + ".");
                NO_Permissao ex = new NO_Permissao("Acesso negado");
                r.exception = ex;
        }
} catch (Throwable t) { // em caso de erro na decisao de acesso, NEGAR
        System.out.println("Error during access decision: access denied.");
        NO_Permissao ex = new NO_Permissao("Acesso negado");
        r.exception = ex;}
}
```

Figura 4.16 – Geração das *Capabilities* pelo Interceptador de Controle de Acesso.

O objeto *RequiredRights*, que contém os direitos requeridos para executar uma determinada operação, é criado estaticamente com os direitos definidos para cada operação da IDL do servidor bancário, conforme apresentado na Tabela 4.5.

O objeto *DomainAccessPolicy* é construído dinamicamente a partir dos direitos atribuídos ao cliente (*gerente* ou *clienteWeb*) por meio de um aplicativo Java (aplicativo *decisão de acesso*). Os direitos podem ser definidos de acordo com as opções apresentadas pelo aplicativo *decisão de acesso*, ilustrado na Figura 4.17. Na Tabela 4.6, tem-se um exemplo de um objeto *DomainAccessPolicy* obtido a partir deste aplicativo.

Descrição	Rights
Registrar novas contas	Jacoweb:**rc**
Apagar conta	Jacoweb:**ac**
Verificar a existência de uma conta	Jacoweb:**ec**
Obter o número de contas do banco	Jacoweb:**nc**
Obter informações de uma dada conta	Jacoweb:**oc**
Efetuar um depósito	Jacoweb:**d**
Efetuar um saque	Jacoweb:**s**
Verificar o saldo	Jacoweb:**vs**
Retirar um extrato	Jacoweb:**e**
Retirar um extrato datado	Jacoweb:**ed**
Efetuar uma transferência	Jacoweb:**t**
Trocar a senha	Jacoweb:**ts**

Tabela 4.5 – Direitos definidos para família *Jacoweb*.

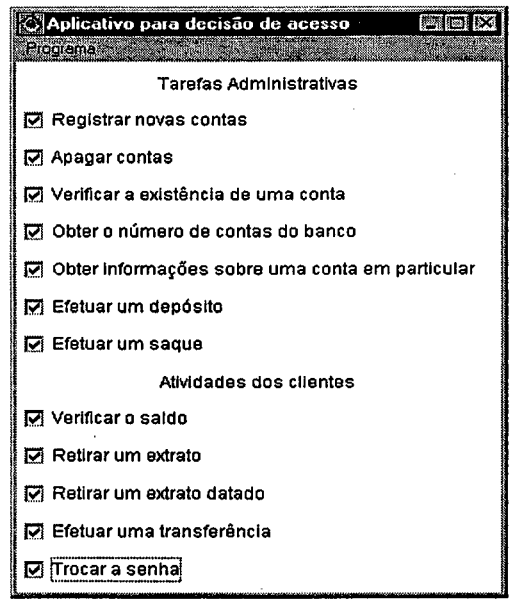


Figura 4.17 – Tela do Aplicativo para Estabelecimento da Política.

Role	Direitos Efetivos (*Granted Rights*)
Gerente	"rc", "ac", "ec", "nc", "oc", "d", "s"
Cliente Web	"vs", "e", "ed", "t", "ts"

Tabela 4.6 – Exemplo de um Objeto *DomainAccessPolicy*.

4.4.4.4 Integração ORB+SSL

A introdução do SSL/TLS como possível mecanismo de segurança no CORBAsec, não prevê o uso de troca de *tokens* de segurança para o estabelecimento de um contexto de segurança (OMG, 2000b). Além disso, a integração de protocolos plugáveis como o SSL ao ORB ainda não está padronizada pelo CORBAsec (OMG, 2000b, RUH, HERRON, *et al.*, 1999). Dessa forma, os

objetos *Vault* e *SecurityContext*, existentes no modelo da figura 4.11, não participam dessa implementação pois sua funcionalidade é substituída pelo protocolo SSL.

Dessa forma, optou-se neste trabalho por uma solução própria que segue o princípio dos objetos de serviço da OMG. O *framework* para a integração do ORB/CORBA com suporte SSL consistiu então em encapsular um pacote SSL pronto (o pacote *iSaSiLk*) na forma de objeto de serviço, tal como os objetos COSS (WANGHAM, 2000). Na Figura 4.18, são apresentados os objetos, com interfaces definidas em IDL, que compõem o pacote *JaCoWeb Security* para a integração ORB+SSL. A classe *Security* encapsula as funcionalidades do pacote *iSaSiLk* e é instanciada pela aplicação no momento de sua criação. O conjunto de parâmetros definidos na criação do objeto *Security* permite ao *iSaSiLk* definir o contexto SSL. O contexto SSL define como cliente e servidor devem selecionar de forma compatível os controles criptográficos (*cipher suite*). A seleção dos controles criptográficos é definida pelos seguintes componentes: método de distribuição de chaves - certificados (*key exchange algorithm*), cifragem para transferências de dados (*symmetric encryption algorithm*) e *message digest* para criação do código de autenticação de mensagem (*hash algorithm*).

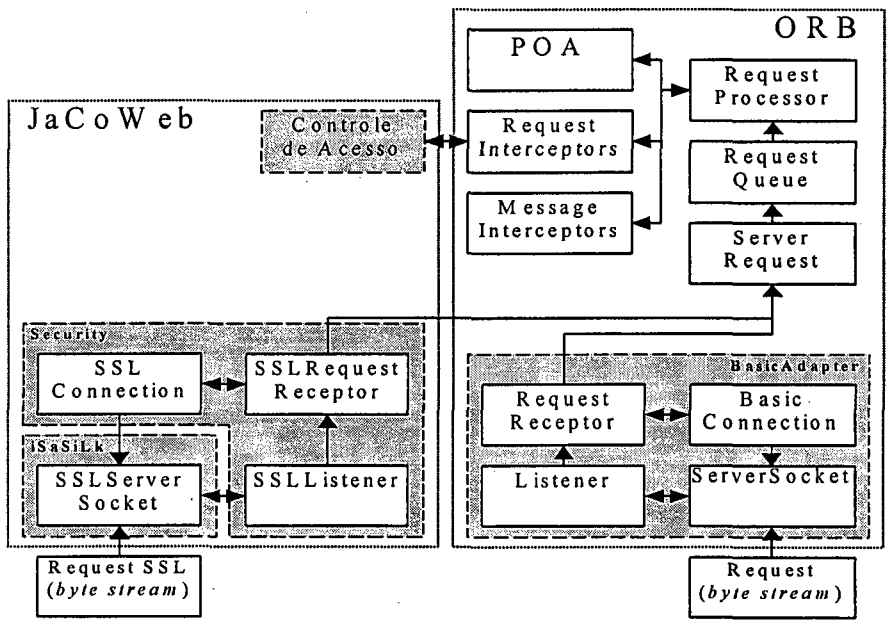


Figura 4.18 – O *framework* ORB+SSL.

Na Figura 4.18, têm-se os objetos *ServerSocket* e *SSLServerSocket*. Um para estabelecer conexões normais através do *Listener* e outro para estabelecer conexões seguras através do *SSLListener*. O *SSLServerSocket* é fornecido pelo iSaSiLk e é encapsulado pelo pacote *JaCoWeb*. O *SSLConnection* tem a função de instanciar o *SSLSocket* para o estabelecimento de uma conexão segura entre cliente e servidor. Portanto, o estabelecimento da chamada segura é realizado dentro

do pacote *JaCoWeb* e, como resultado, o conjunto de *byte stream* da requisição cliente é passado para o ORB através do *SSLRequestReceptor*. Convém notar que ambos *Listeners* encaminham o conjunto de *byte stream* para o seu respectivo objeto receptador de requisição (*RequestReceptor* e *SSLRequestReceptor*, respectivamente) que realiza a conversão para remontar o objeto *Request*. A partir desse momento, a requisição pode seguir seu caminho normal até o POA, onde ocorre a transformação dos parâmetros do objeto *Request* em argumentos a serem passados ao *servant*.

Neste experimento, o interceptador de chamada segura deixa de ter uma função mais ativa no *framework*, e é utilizado somente para obter informações sobre o contexto de segurança do objeto SSL. No entanto, pode também ser utilizado para fins relacionados com a aplicação.

O ponto chave para esta integração dar certo é definir como a aplicação pode selecionar o modo de conexão a ser utilizado (segura ou convencional). No modelo CORBA, um objeto pode ser localizado nos sistemas distribuídos através da sua referência de objeto. Na OMG, a referência de objeto é definida numa forma padrão conhecida como IOR (*Interoperable Object Reference*). A IOR (OMG, 1999b) é uma seqüência de caracteres que, quando convertida para o formato adequado, fornece as informações de endereço IP do sítio, da porta, de um ponteiro para o objeto a ser acessado e algumas informações de controle relacionadas à própria IOR. Cada objeto distribuído deve ter sua IOR disponível, registrada em um serviço de nomes. A especificação CORBAsec define uma extensão da IOR de forma que possa agregar informações específicas para que o ORB trate também de conexões seguras. A extensão da IOR é feita adicionando um novo tipo de *Tag* (etiqueta) de segurança ao corpo do IIOP *profile* da IOR[24].

Para estabelecer uma associação segura utilizando o SSL, o servidor da aplicação deve registrar-se no Serviço de Nomes com uma IOR que contenha a *Tag* SSL chamado SSLIOP (OMG, 2000a). Para isto, modificações no método *ORB.createIOR ( )* do JacORB foram necessárias. E, como os clientes necessitam conhecer a porta SSL e os requisitos necessários para estabelecer a conexão segura, a classe *ParsedIOR*, responsável pela decodificação da IOR, também foi alterada.

Vale ressaltar que, apesar do servidor ter disponível duas portas de acesso (uma segura e outra convencional), qualquer tentativa indevida de acesso a um serviço (objeto) com restrições de segurança pela porta convencional é inibida pelo serviço de controle de acesso no interceptador de requisição (Figura 4.18).

Quando as aplicações desejam estabelecer associações seguras utilizando o *framework* ORB+SSL, algumas informações de segurança precisam ser inicializadas. O objeto *Security* é o responsável pela iniciação da segurança, conforme definido em sua interface IDL (ver Figura 4.19).

[24] Os *tagged profiles* foram discutidos na seção 3.4.8.

Como descrito na seção 4.4.4.1, após a inicialização do ORB e do POA (somente para servidores), as aplicações devem completar a inicialização do ORB seguro. Nesta implementação deve-se instanciar um objeto *Security*, de acordo com o seu tipo de aplicação (cliente *stand-alone*, *applet* ou servidor).

Para o *applet* cliente, as informações de segurança são definidas através do primeiro método da interface IDL, apresentadas na Figura 4.19 (*initAsClient* (orb,applet)). Este método instancia um objeto *Security* e invoca o método *initAsClient* (orb). Já para aplicações *standalone*, o contexto de segurança pode ser definido através do método initAsClient (orb), que deve: a partir do certificado do cliente definir o principal; criar a credencial SSL estática do cliente (com base no certificado deste cliente); criar e definir os objetos de política utilizados no estabelecimento da associação segura; definir o contexto SSL do cliente, especificando os *cipher suites*, o objeto *TrustDecider* (responsável pela autenticação do servidor) e o certificado do cliente; e, finalmente, definir estas configurações de segurança (objeto *Security*) na classe jacorb.orb.Environment.

```
// $Id: Security.idl, v1.0 1999-12-08
#include "orb.idl"
#pragma prefix "edu.lcmi"
module jacoweb {
  interface Security {
            Security initAsClient ( //inicializacao do cliente (applet)
              in ORBorb,
              in Applet          applet
            );
            Security initAsClient ( //inicializacao do cliente (stand-alone)
              in ORBorb,
            );
            Security initAsServer ( //inicializacao do servidor
              in ORBorb,
              in PortableServer::POA poa
            );
  }
}
```

Figura 4.19 – Interface IDL do pacote *JaCoWeb*.

4.4.5 Testes do Protótipo

Os testes do protótipo foram feitos no âmbito da avaliação da segurança do Esquema de Autorização *JaCoWeb*. Esta avaliação é detalhada no capítulo 6 e **um dos pontos chave para a garantia da segurança** de um sistema são os **testes** (ISO/IEC 15408-1, 1999, ISO/IEC 15408-2, 1999, ISO/IEC 15408-3, 1999).

Testar é um método dinâmico para verificação e validação onde o sistema a ser testado é executado e seu comportamento é observado (JALOTE, 1991). São empregados freqüentemente diferentes níveis de testes. O **teste de unidade** é usado para testar um módulo ou um número pequeno de módulos. Seu objetivo é detectar erros na codificação dos módulos. Durante o **teste de integração**, módulos são combinados em subsistemas que são então testados. A meta é testar o

projeto do sistema. No **teste de sistema** e **teste de aceitação**, o sistema inteiro é testado. O objetivo é testar o sistema de acordo com os requisitos, e testar os próprios requisitos.

Existem duas **abordagens** de testes: **funcional** e **estrutural**. No teste funcional, não é considerada a lógica interna do sistema a ser testado e os *testcases* são escolhidos a partir das especificações ou dos requisitos. É chamado freqüentemente "teste de caixa preta". O particionamento de classes de equivalência, a análise de valores limite e os diagramas de causa-efeito são exemplos de métodos para selecionar *testcases* para os testes funcionais (JALOTE, 1991). O teste estrutural se preocupa em testar a implementação do *software*. Os testcases são decididos completamente a partir da lógica interna do programa/módulo que está sendo testado. Embora um critério estrutural possa ser especificado, o procedimento para selecionar *testcases* é de responsabilidade do testador. É chamado freqüentemente de "teste de cobertura". Os critérios estruturais mais comuns são: cobertura de declarações (execução de todas as declarações do sistema) e, cobertura de ramos (execução de caminhos lógicos do sistema).

Testcases apropriados são **pontos cruciais para um teste de sucesso**. Embora todas as faltas em um programa não possam ser reveladas pelos testes e, devido a limitações econômicas, a meta de seleção de *testcases* é selecionar *testcases* tal que o máximo número possível de faltas seja descoberto pelo número mínimo possível de *testcases*. Seleção de *testcases* ainda não é um simples processo mecânico. Conhecimento e criatividade do testador ainda são importantes, apesar da disponibilidade de ferramentas que selecionam *testcases* para assegurar cobertura.

Normalmente os testes começam com a definição de um **plano de teste** que é o documento guia básico para o teste inteiro do *software*. Esse plano especifica os níveis de teste e as unidades a serem testadas. Para testar unidades diferentes, os *testcases* são especificados e então executados.

Para obter resultados de **testes no protótipo do *JaCoWeb***, foi definido um **plano de teste**, disponível em (http://www.lcmi.ufsc.br/~merkle/TestPlan.html). Dois tipos de testes foram executados: **testes de unidade** e **testes de sistema**. As unidades testadas foram os objetos de aplicação (*applet* de aplicação, servidor bancário e servidor de nomes) e os objetos de serviço do CORBAsec - *AccessDecision*, *RequiredRights*, *DomainAccessPolicy* e *Current*. *Testcases* estão disponíveis em (http://www.lcmi.ufsc.br/~merkle/TestCases.html).

Os **testes de unidade** foram baseados em **cobertura de ramos** e o objetivo foi cobrir 95% dos ramos lógicos do protótipo. Testes de unidade foram aplicados a cada um dos objetos de serviço do CORBAsec listados acima. Um diagrama UML foi feito para cada um deles. Um **relatório de teste de unidade** está disponível em (http://www.lcmi.ufsc.br/~merkle/UnitTesting.html).

Testes de sistema usaram **conjuntos de valores válido e inválidos** (particionamento de classes de equivalência), **valores limites, conjunto de valores especiais** e **diagramas de causa-efeito** para definir *testcases*. Testes de sistema foram aplicados para o *applet* de aplicação, o servidor bancário e os objetos de serviço do CORBAsec que lidam com controle de acesso e tarefas de chamada segura. Foram projetados **diagramas *usecases* UML** para descrever os objetos de aplicação e **diagramas de classe UML** para descrever a funcionalidade dos objetos de serviço do CORBAsec. Um **relatório de teste de sistema** está disponível em (http://www.lcmi.ufsc.br/~merkle/SystemTesting.html).

4.5 Trabalhos Correlatos

A literatura apresenta alguns trabalhos sobre gerência de políticas de segurança e uso de *capabilities* em ambientes de larga escala.

No que diz respeito à gerência de políticas de segurança, a própria especificação do serviço de gerência de domínios (OMG, 2000f) do CORBAsec não está padronizada. Um projeto acadêmico que pode ser mencionado é o ***Cherubim*** (CAMPBELL and QIAN, 1998). O ***Control*** (ADIRON, 1999) é um produto que implementa o CORBAsec e a gerência de políticas de segurança.

A arquitetura de segurança do projeto *Cherubim* consiste de duas partes: uma IDL CORBA estendida com o objetivo de automatizar a *ligação* entre políticas de segurança e aplicações, e um *framework* de segurança para agentes dinâmicos. Seu principal objetivo é construir um sistema de segurança dinâmico capaz de prover mecanismos de segurança específicos para aplicações em um ambiente móvel e de larga escala. O controle de acesso nesse projeto foi desenvolvido usando-se o conceito de *capabilities*, que carregam os direitos de acesso do *principal* até a máquina do servidor onde ocorre o processo de autorização. O projeto elaborou interfaces para manipular políticas de autorização discricionárias, obrigatórias (*MAC*) e baseadas em papéis (*RBAC*), mas não utiliza os objetos *COSS* do *CORBAsec* implementando objetos próprios. O ORB utilizado é o *JacORB* (BROSE, 1997) e os serviços criptográficos são implementados com a API criptográfica do *Java IAIK* (GRAZ, 1999b).

O *Control* (ADIRON, 1999) é um ORB que estende o ORB *ORBAsec* com os serviços *COSS* do *CORBAsec* que implementam serviços de autenticação, transmissão segura de mensagens e controle de acesso automático. A política de autorização discricionária é estabelecida com uma linguagem de controle de acesso usada por objetos servidores para definir seus objetos de política. A interceptação do controle de acesso é executada no lado do objeto servidor. A gerência de políticas de segurança é feita usando-se o módulo de gerência de domínios.

Comparando o *PoliCap* com as experiências apresentadas, verificamos que o *PoliCap* é um serviço que realiza o controle de acesso *no lado do cliente*, ao contrário das propostas descritas. O *PoliCap* combina características do *Cherubim*, como as *capabilities*, e do *Control*, com o adicional do uso restrito de objetos padronizados ou em via de padronização no CORBAsec.

Em **outros tipos de trabalhos** relacionados com políticas de autorização em ambientes de larga escala que usam o CORBAsec (ADIRON, 1999, BROSE, 1998, 2000, LANG, 1999), o principal objetivo é propor um novo modelo teórico de controle de acesso, diferente do modelo padronizado no CORBAsec. Nesses trabalhos, a política de autorização é expressa através de uma linguagem própria, e nenhum enfoque é centrado no gerenciamento dos objetos de política no CORBAsec, embora Gerald Brose (BROSE, 2000) cite este problema como perspectiva futura em seu trabalho.

Vários trabalhos existentes na literatura como o **sistema Oasis** (BACON, MOODY, *et al.*, 2000, HAYTON, BACON, *et al.*, 1998), o **projeto Racoon** (BROSE, 2000), o **sistema I-CAP** (GONG, 1989) e o **sistema de proteção de** (HAGIMONT, HUET, *et al.*, 1997) utilizam ***capabilities*** em seus sistemas de segurança, como já foi mencionado na seção 4.3. Porém, essas *capabilities* representam, em um sentido tradicional, os direitos pertencentes a um *principal* ou requisição que possibilita o acesso a um objeto, e são estabelecidos ou de forma estática, em (BROSE, 2000, HAGIMONT, HUET, *et al.*, 1997), ou de forma dinâmica em (BACON, MOODY, *et al.*, 2000, GONG, 1989, HAYTON, BACON, *et al.*, 1998). A vantagem do uso de *capabilities* é a sua flexibilidade em permitir a evolução dinâmica dos direitos de acesso no sistema (HAGIMONT, HUET, *et al.*, 1997). Na nossa proposta e implementação, as *capabilities* são determinadas de forma dinâmica a cada requisição, representando os direitos de acesso de cada pedido. Também reduzem o tráfego de rede em caso de negação do pedido de acesso, sendo verificadas no lado do servidor.

4.6 Considerações Finais

É necessário que se faça algumas considerações sobre o **Esquema de Autorização *JaCoWeb* Discricionário** implementado (WANGHAM, LUNG, *et al.*, 2000), de acordo com os requisitos levantados na especificação CORBASec (OMG, 2000a) e nas considerações dessa especificação apresentadas em (LANG e SCHREINER, 2000). Os requisitos, nos quais estão baseadas estas considerações são: transparência, simplicidade, reusabilidade, escalabilidade, flexibilidade e interoperabilidade.

A **transparência** oculta as funcionalidades (mecanismos) do sistema de segurança do usuário e do programador da aplicação, dando a impressão de que as invocações são atendidas

localmente. O aspecto da **simplicidade** indica que o sistema de segurança deve ser fácil de entender, usar, e gerenciar. O *JaCoWeb Security* tenta atender a esses dois requisitos, da melhor maneira possível, definindo um objeto *Security* responsável pela inicialização e ativação de todos os mecanismos de segurança. A partir daí, as invocações são realizadas tal como num ORB convencional. No aspecto da facilidade de gerenciamento das configurações e políticas de segurança, o *JaCoWeb* é implementado de forma que as configurações de sistema sejam definidas estaticamente, em tempo de compilação, na classe *Environment*. No entanto, as políticas de segurança, para o controle de acesso às aplicações, podem ser definidas dinamicamente em tempo de execução através da interface de gerenciamento apresentada.

A **reusabilidade** é uma consideração importante, já que diminui o tempo de desenvolvimento e manutenção dos programas, tornando-os mais robustos e confiáveis. A infra-estrutura de segurança do *JaCoWeb*, por ser baseada no modelo de objetos de serviço, pode ser facilmente adaptada para diferentes aplicações. Todos os serviços são descritos em IDL padrão OMG, o que contribui não só no aspecto da reutilização de código como também para a portabilidade.

O aspecto da **escalabilidade** diz respeito à possibilidade do sistema de segurança se adequar tanto para sistemas pequenos ou amplos, quanto ao número de usuários e operações, isto é, que tenha suporte para domínios de políticas, grupos, *roles*, etc. A implementação desenvolvida usa conceitos de grupo e *roles* dentro de um domínio de política único, de maneira simplificada.

Em (LANG e SCHREINER, 2000) é discutida a relação entre os requisitos de **flexibilidade** e **interoperabilidade**. A máxima flexibilidade é alcançada quando o número de interfaces e protocolos padronizados é o menor possível, permitindo maior liberdade aos desenvolvedores e usuários para estender os mecanismos de segurança de acordo com suas necessidades. De modo contrário, a máxima interoperabilidade é alcançada quando se tem um grande número de interfaces e protocolos padronizados. Isto assegura que ORBs de diferentes companhias ofereçam as mesmas funcionalidades, implicando na possibilidade de aplicações, em ORBs distintos, possam interagir.

É claro que existe a dificuldade em atender completamente a cada um desses requisitos sem detrair o outro. O *JaCoWeb Security* se utiliza das opções que a especificação CORBASec oferece para atender, de forma equilibrada, esses requisitos. Novamente, o uso de interfaces IDL e do modelo de objeto de serviço (COSS) na implementação do *JaCoweb Security* nos permitiu atender melhor o aspecto da interoperabilidade. No entanto, como as especificações CORBASec padronizam uma grande parte de todas as abstrações de segurança (mecanismos, propriedades, políticas, estrutura de dados, etc.), o aspecto da flexibilidade é bastante afetado. Contudo, isto não

impede que extensões proprietárias sobre o modelo sejam implementadas na forma de objeto de serviço – sem alterar as características do ORB em si.

Em relação a **comunicação**, no estabelecimento do contexto de segurança através das tecnologias de segurança subjacentes, o *JaCoWeb Security* adota o SSL juntamente com os protocolos de adaptação SSLIOP (*Secure Socket Layer Inter-ORB Protocol*) e SIOR (*Security-enhanced Inter-ORB Reference*), padrões da OMG. Isto significa que a interoperabilidade com outros ORBs só é alcançada quando ambos utilizam os mesmos protocolos (SSLIOP) e tenham políticas de segurança "consistentes" (LANG e SCHREINER, 2000). No entanto, o *framework JaCoWeb Security* foi modelado de forma a permitir uma fácil adaptação a outras tecnologias de segurança, desde que tenha uma API disponível.

4.7 Conclusões do Capítulo

O *PoliCap* preenche uma lacuna no gerenciamento de políticas de segurança existente no modelo CORBAsec, concretizando o primeiro nível de controle de acesso do projeto *JaCoWeb*. O segundo nível de controle de acesso é desenvolvido com o uso das *capabilities* propostas para o modelo CORBAsec. O *PoliCap* e as *capabilities* para o CORBAsec fornecem uma contribuição importante para gerenciar políticas de autorização em redes de larga escala. Importante também é a descrição do funcionamento dinâmico relativo aos objetos de serviço e interceptadores do CORBAsec, no contexto de uma aplicação real como o protótipo desenvolvido.

Conclui-se nesse trabalho que a integração ORB+SSL não deve ser implementada no núcleo do ORB, para que este não perca características importantes como interoperabilidade. Para que essa integração não perca a aplicabilidade em sistemas abertos, propõe-se que o objeto definido como substituível, o *Vault*, manipule a negociação do contexto de segurança da associação (*handshake* do SSL). Dessa forma, as implementações do núcleo do ORB e dos serviços ORB estarão independentes da tecnologia de segurança subjacente.

O protótipo é uma implementação do modelo discricionário do CORBAsec, considerando que existem poucas experiências desse tipo disponíveis, pela falta de experiência de implementação da especificação CORBAsec (ALIREZA, LANG, *et al.*, 2000). Esse protótipo facilita a gerência e administração de políticas de segurança, permitindo que a segurança seja garantida a nível de ORB. A segurança garantida a nível de ORB tem vantagens como transparência da segurança para aplicações e maior confiança nos serviços de segurança que são sempre executados.

O protótipo implementado visou fornecer segurança a uma aplicação crítica muito utilizada pelos navegadores da Internet. Um sistema *internet banking* exige que somente usuários autorizados poderão ter acesso aos serviços do sistema e que as informações trocadas entre os

clientes do banco e o servidor bancário necessitam ser criptografadas de forma que nenhum "bisbilhoteiro" tenha acesso a essas informações.

As tecnologias e ferramentas empregadas neste trabalho foram escolhidas devido à forte aceitação na Internet (SSL e o pacote iSaSiLk, linguagem Java e o JDK 1.2.1) e em ambientes CORBA (JacORB).

O protótipo do esquema de autorização discricionária garante à aplicação crítica selecionada boa parte das características desejáveis de segurança, tais como autenticidade, integridade, confidencialidade e uso legítimo. Apesar das dificuldades para seguir o modelo CORBAsec, devido às abstrações desta especificação, o protótipo implementado atingiu o seu objetivo, que era comprovar a viabilidade e a confiabilidade do esquema de autorização proposto.

A implementação dos objetos definidos no *PoliCap* irão levar o nosso modelo para o seu formato original, onde os objetos locais são obtidos a partir de objetos globais, concretizando o primeiro nível de autorização do esquema *JaCoWeb*, o que representa uma perspectiva futura para o projeto *JaCoWeb Security*.

Espera-se que a descrição da aplicação do modelo CORBAsec no *software JaCoWeb Security* auxilie na implementação de aplicações práticas em ambientes de objetos distribuídos, que utilizam ORBs seguros, segundo o modelo CORBAsec, e desperte a comunidade científica para o desenvolvimento dessa linha de pesquisa ainda aberta a modificações e melhorias.

Nesta versão do protótipo implementamos apenas políticas discricionárias. Políticas não-discricionárias ou obrigatórias são propostas no capítulo 5.

Esse protótipo, com os testes realizados, forneceu subsídios para avaliação com relação aos Critérios Comuns de segurança (ISO 15408), detalhada no capítulo 6.

Capítulo 5
ESQUEMA DE AUTORIZAÇÃO OBRIGATÓRIO

5.1 Introdução

Durante as três últimas décadas, órgãos militares americanos investiram recursos consideráveis na pesquisa e desenvolvimento da segurança computacional, na tentativa de encontrar formas de assegurar a comunicação eletrônica sobre suas redes privadas. Com o "desespero" comercial que existe atualmente quanto a segurança em ambientes de redes, parece óbvio procurar soluções nas tecnologias geradas em ambientes militares e aplicá-las no âmbito da Internet (DALTON e GRIFFIN, 1997).

Em ambientes de organizações públicas ou privadas, onde a descoberta de informações sigilosas por pessoas não autorizadas pode comprometer a própria existência destas organizações, há a necessidade de alta garantia de confidencialidade no fluxo de informações nestes ambientes. Isto, portanto, motiva o uso de modelos de segurança obrigatórios.

Várias políticas obrigatórias fazem parte do dia-a-dia em nossa vida. Por exemplo, a diretoria de uma empresa pode fornecer o direito a um funcionário de consultar o seu salário, mas não deseja que esse funcionário passe este direito para outro funcionário. De forma mais geral, as pessoas freqüentemente fornecem a uma segunda parte interessada (talvez um médico ou advogado) o direito de conhecer informações pessoais sobre as mesmas, mas apenas com a condição de que esse direito não seja passado para uma terceira parte arbitrária qualquer.

As características das políticas obrigatórias, como a possível concretização do princípio do mínimo privilégio (*least privilege*) e a existência de controles obrigatórios, sobre os quais os usuários não tem controle, auxiliam a controlar as violações de segurança existentes em aplicações comerciais complexas que executam em larga escala (ZHONG e EDWARDS, 1998). A política obrigatória torna possível prover ambientes restritos que encapsulam um componente COTS (*Commercial-Off-The-Shelf*) do sistema, de acordo com o seu comportamento, sem a necessidade de verificações no seu código fonte. No trabalho desenvolvido por Qun Zhong e Nigel Edwards (ZHONG e EDWARDS, 1998), criou-se um conjunto de programas confiáveis, usando políticas obrigatórias, que encapsulam a aplicação *Sendmail*[25] do ambiente.

[25] Um dos programas mais comuns usados para fornecer serviços de correio eletrônico na Internet. É um dos alvos de ataques favoritos de intrusos na Intenet.

A política discricionária, na qual os direitos de acesso a cada recurso são manipulados livremente pelo responsável do recurso, segundo a sua vontade, possui uma falha evidente: esta não possui nenhuma proteção contra os "cavalos de Tróia" (MCLEAN, 1994). As políticas obrigatórias permitem estabelecer regras incontornáveis que asseguram a proteção eficaz contra a descoberta ou fuga de informações.

Em (GLADNEY, 1997) afirma-se que é necessário existir uma síntese das necessidades discricionárias e obrigatórias em ambientes tanto comerciais como militares, para se ter um controle de acesso adequado para aplicações de larga escala.

O modelo mais citado que guia a construção de políticas obrigatórias é o modelo **Bell e Lapadula**, ou modelo BLP (BELL e LAPADULA, 1976, LAPADULA e BELL, 1996). Embora exista um grande número de críticas ao uso do Bell e Lapadula em ambientes computacionais (CLARK e WILSON, 1987, FOLEY, 1994, LANDWEHR, HEITMEYER, *et al.*, 1984, MCLEAN, 1987, 1994, SANDHU, 1993), atualmente, vários pesquisadores propõem formas de suavizar os problemas encontrados neste modelo (KARJOTH, 1998, 2000, NICOMETTE, 1997, OSBORN, SANDHU, *et al.*, 2000).

Nosso objetivo é usar o esquema de autorização *JaCoWeb* para concretizar o modelo Bell e Lapadula definindo então políticas obrigatórias em ambientes distribuídos. O **esquema de autorização *JaCoWeb* Obrigatório**, descrito neste capítulo, visa:

- Descrever as **técnicas** utilizadas pelo esquema obrigatório para **suavizar** o problema da **superclassificação** do modelo Bell e Lapadula, descrito no capítulo 2, que representa a principal restrição ao uso deste modelo (seção 5.2);
- Definir uma **extensão** do modelo de **segurança** do **CORBA**, que não prevê o uso de políticas obrigatórias, adotando o modelo Bell e Lapadula para definir um controle **obrigatório** a partir do serviço de política ***PoliCap*** (seção 5.3);
- Definir as **regras** do esquema de autorização obrigatório e definir como será realizada a **evolução dos rótulos** de segurança no sistema (seção 5.3);
- Definir a forma do **estabelecimento** e do **uso** de **políticas obrigatórias** no Esquema de Autorização *JaCoWeb* Obrigatório (seção 5.3).

5.2 Considerações sobre o Modelo Bell e Lapadula

Nesta seção serão feitos alguns comentários sobre a modificação dos rótulos de segurança e sobre o problema da superclassificação no modelo Bell e Lapadula.

5.2.1 Modificação dos rótulos de segurança no Modelo Bell e Lapadula

O **princípio da tranqüilidade** (LAPADULA e BELL, 1996) estabelece que os rótulos de segurança e os direitos de acesso nunca são modificados no modelo BLP. Entretanto, implementações baseadas no modelo normalmente utilizam algum meio para modificar esses rótulos de segurança e/ou os direitos de acesso, de forma a tornar prática a implementação em um sistema computacional.

A dinâmica do modelo Bell e LaPadula define como o **nível corrente** de segurança de um **sujeito** evolui, ou pode evoluir, e, também, como os **rótulos de segurança** dos **objetos** podem ser modificados, ambos durante a operação do sistema (BELL e LAPADULA, 1976). A dinâmica do modelo BLP é um assunto bastante discutido na literatura (AMOROSO, 1994, FOLEY, 1998, FOLEY, GONG, *et al.*, 1996, LANDWEHR, 1981, MCLEAN, 1990, NICOMETTE, 1996, SANDHU, 1993, TIPTON, 1998).

Quando um usuário entra no sistema, ele recebe um nível corrente de segurança f_C que seja dominado pela sua habilitação. Este rótulo pode ser escolhido pelo usuário ou pode ser atribuído automaticamente pelo sistema. Os sujeitos criados em nome de um usuário herdam tanto a habilitação como o nível corrente de segurança do usuário. Os acessos destes sujeitos aos objetos do sistema devem observar a propriedade-ss e a propriedade-*.

A regra número 10 do modelo BLP, chamada *change-subject-current-security-level*, dita que o rótulo corrente de segurança de um sujeito só é modificado mediante uma requisição explícita deste sujeito; isto significa que o rótulo corrente de segurança não flutua de maneira automática no sistema, e também que qualquer flutuação ocorre por iniciativa do próprio sujeito. A regra especifica também, como condições, que a alteração do rótulo corrente de segurança só é autorizada se ela não violar a propriedade-*, e se o rótulo corrente for dominado pela habilitação do sujeito. Respeitando-se essas condições, o rótulo corrente pode, portanto, subir ou descer no reticulado de rótulos de segurança. Nada impede que a implementação do modelo modifique esse rótulo de forma automática, desde que respeite as condições impostas pela regra.

Por exemplo, seja a seguinte situação: um sujeito s, com rótulo corrente f_C = NÃO-CLASSIFICADO e habilitação f_S = SECRETO, deseja ler um objeto $o1$, que possui f_{O1} =

CONFIDENCIAL. A propriedade-ss não permite que *s* leia *o1*, pois $f_{o1} \not\geq f_C$. Logo, *s* precisa solicitar a atualização de seu rótulo corrente f_C para (pelo menos) CONFIDENCIAL.

O modelo BLP possui também a regra número 11, chamada *change-object-security-level*, que dita que o rótulo do objeto pode ser modificado mediante uma requisição explícita de um sujeito. Entretato, também existem condições impostas pela regra para que essa modificação possa ocorrer respeitando as propriedades do modelo (BELL e LAPADULA, 1976).

Um exemplo prático e bastante sutil da dinâmica da modificação de rótulos de segurança, pode ser representado pelo seguinte trecho em pseudocódigo (MCLEAN, 1990):

```
1  passa(A1: arquivo, A2: arquivo)
2  abre A1 para leitura
3  lê I de A1
4  fecha A1
5  abre A2 para escrita
6  escreve I em A2
7  fecha A2
```

Considerando os arquivos A1 e A2, com rótulos de segurança iguais a SECRETO e CONFIDENCIAL, respectivamente, como os *objetos* do sistema e, a implementação do código *passa* como *sujeito* do sistema. Dessa forma, esse código pode servir de base para uma revelação não-autorizada. Entre as linhas 2 e 4, *passa* tem que ter rótulo corrente SECRETO, para que possa ter acesso de leitura à A1. Entretanto, nada (a princípio) impede que entre as linhas 4 e 5 ele rebaixe seu rótulo corrente para CONFIDENCIAL, o que lhe permite escrever a informação I no arquivo A2 (linhas 5 a 7). Isto representa um fluxo não-autorizado de informação, embora a propriedade-* não esteja sendo violada. Situações como esta não devem ser percebidas como uma limitação do modelo Bell e LaPadula, mas sim, como uma restrição imposta pelas condições de implementação do modelo.

5.2.2 Técnicas para evitar a superclassificação do Modelo BLP

A propriedade-*, no exemplo de cópia de um arquivo classificado como CONFIDENCIAL, por um sujeito que possui o f_C = SECRETO, na seção 2.5.2.1, impõe que a classificação da cópia deve ser superior ou igual ao nível corrente de segurança do sujeito, caracterizando a subida da informação no reticulado de rótulos de segurança e o fenômeno conhecido como **superclassificação da informação** (LANDWEHR, 1981, WOODWARD, 1987). Este fenômeno resulta do aspecto extremamente restritivo da propriedade-*, e representa, como já foi dito, um dos principais problemas deste modelo.

Algumas técnicas são sugeridas na literatura para evitar ou suavizar a superclassificação das informações no modelo BLP (BELL e LAPADULA, 1976, LANDWEHR, 1984, FOLEY, 1998, FOLEY, GONG, *et al.*, 1996, NICOMETTE, 1996, WOODWARD, 1987).

Sujeitos e Intervalos de Confiança

O próprio documento original do BLP (BELL e LAPADULA, 1976), sugere o uso de **sujeitos de confiança** (*trusted subjects*), que não precisam respeitar a propriedade-* do modelo, a fim de possibilitar a realização de leituras e escritas livres de quaisquer restrições. Já em (BELL, 1986), um dos próprios projetistas do modelo original estabelece que os sujeitos de confiança podem violar a propriedade-*, mas de forma controlada. Para atingir este objetivo, um par de rótulos de segurança L_r e L_w é associado a cada sujeito. A propriedade-ss é aplicada considerando-se o rótulo L_r e a propriedade-* é aplicada considerando-se o rótulo L_w. É necessário que L_r domine L_w, ou seja, $L_r \geq L_w$. Com esta restrição, o sujeito pode ler e escrever em um intervalo de rótulos entre L_r e L_w que é chamado **intervalo de confiança** (*trusted range*).

A superclassificação da informação provoca a necessidade de reclassificações periódicas dos objetos, que pode ser feito através dos sujeitos de confiança, apenas para garantir a usabilidade de sistemas baseados no modelo BLP.

Na extensão do BLP descrita em (LANDWEHR, 1984), existe a possibilidade da realização da operação *downgrade* no sistema. Essa operação permite o "rebaixamento" da classificação de um objeto a um usuário que assume o papel (*role*) denominado *downgrader*, que normalmente é assumido por usuário autorizados pelo administrador do sistema.

Processos confiáveis / Objetos sem estado

O conceito de **processos confiáveis** (*trusted processes*) foi introduzido como uma implementação do conceito de sujeitos de confiança (estendida aos objetos), e foi usado por vários sistemas que implementaram o modelo BLP como: o MME (*Military Message Experiment*), o Multics AFDSC (*Air Force Data Services Center*) e o KSOS (*Kernelized Secure Operating System*) (LANDWEHR, 1984). Por exemplo, o administrador do sistema pode executar processos confiáveis relacionados com a manutenção do sistema, para conseguir realizar suas tarefas administrativas sem restrições. Outra classe de processos que pode ser considerada confiável compreende os subsistemas mais críticos do sistema operacional, como gerência de memória e *drivers* de dispositivos. O trabalho de (NICOMETTE, 1996) considera confiáveis processos ou objetos que ele denomina **objetos sem estado**. Aos objetos sem estado, a exemplo da abordagem sugerida por (BELL, 1986) para os sujeitos confiáveis, são atribuídos intervalos de confiança [Lmin, Lmax]. Lmin representa o nível de segurança **mínimo** dos dados que um objeto sem estado

pode **escrever**. Lmax representa o nível de segurança **máximo** que pode ser **lido** pelo objeto sem estado.

Todo objeto é capaz de conservar dados no seu espaço de endereçamento mas os dados podem ser classificados em duas categorias: dados do sistema e dados de aplicativos. Os dados do sistema são as informações que são manipuladas pelo sistema para que todo processo possa executar. Os dados de aplicativos são dados que compõem o estado do objeto do ponto de vista do implementador da aplicação, por exemplo, os dados declarados pelo programador no código da aplicação.

Um objeto **sem estado** é um objeto que não guarda nenhum dado de aplicativo na memória entre duas ativações sucessivas[26]. Seu estado (ou o estado da aplicação) é reinicializado a cada invocação do objeto. Essa propriedade dos objetos sem estado é muito importante pois significa que duas requisições sucessivas que acessam o mesmo objeto sem estado **não podem realizar um fluxo de informação** (não podem trocar informações) por meio desse objeto. De forma contrária, um objeto **com estado** armazena dados de aplicativos e seus diferentes métodos consistem na consulta ou modificação desses dados. Um arquivo é um bom exemplo de objeto com estado.

A noção de objeto com estado é natural a todo programador ou aos programadores de linguagens orientadas a objetos enquanto a noção de objeto sem estado pode causar certa confusão. Um bom número de exemplos de objetos sem estado podem ser encontrados nos sistemas atuais. Por exemplo, servidores de arquivo NFS (*Network File System*) que possibilitam a criação de um sistema de arquivos distribuído de forma transparente é um objeto sem estado. Esse objeto responde às requisições de montagem de sistemas de arquivos sem guardar nenhuma informação sobre o atendimento de uma requisição a ser usada na próxima requisição. Da mesma forma, **processos servidores do sistema** em geral (os navegadores, os servidores de impressão, por exemplo) são objetos **sem estado**. Por exemplo, considerando um ambiente bancário, um objeto *Conta_bancária* armazena o estado e dados sobre as contas dos clientes no seu espaço de endereçamento, podendo ser considerado um objeto com estado. Já um objeto *Servidor_de_Impressão* apenas tem a informação sobre o objeto a ser impresso no momento de sua impressão, para depois permanecer sem informação nenhuma sobre o objeto impresso. O *Servidor_de_Impressão* não guarda nenhuma informação em memória entre duas requisições sucessivas. Essa distinção de objetos com estado e sem estado possibilita obter um esquema de autorização menos restritivo do que o Bell e Lapadula.

[26] A origem do conceito dos objetos sem estado é fundamentada no conceito de *Object Reuse*, definida pelo TCSEC em (DOD, 1985). *Object reuse* é definido como a reutilização, feita por um sujeito, de algum meio de armazenamento (setor de disco rígido, página de memória, etc), que armazenou um ou mais de um objeto. Para ser reutilizado de forma segura, nenhum dado residual pode ficar disponível ao novo sujeito, quando este reutiliza o objeto.

Sensibilidade Inicial dos Objetos e Rótulos Flutuantes

No trabalho clássico de (WOODWARD, 1987), o autor explica que o problema da superclassificação é ocasionado porque, em implementações tradicionais de sistemas baseados no BLP como o SCOMP (FRAIM, 1983), os sujeitos e objetos **herdam o nível de segurança do seu criador**. Levando em consideração que um objeto recém criado não contém dado algum, associar o nível de segurança do seu criador imediatamente não representa o nível de sensibilidade real das informações contidas neste objeto naquele momento. O autor sugere **inicializar** qualquer **objeto recém criado** com f_O = **NÃO-CLASSIFICADO**.

Durante a execução de quaisquer operações subseqüentes, o nível de segurança do objeto deve "flutuar" para representar o nível de sensibilidade das informações que são armazenadas em seu interior. Um sistema que implementa rótulos flutuantes suaviza os problemas de superclassificação inerente aos sistemas tradicionais (WOODWARD, 1987).

A modificação de rótulos de segurança feita de forma dinâmica pelo próprio sistema, que tem como objetivo evitar a superclassificação da informação usando as técnicas mencionadas acima, habilita um sistema a atender uma variedade de requisitos de segurança muito maior do que os modelos onde os rótulos são estáticos (FOLEY, GONG, *et al.*, 1996, NICOMETTE, 1996, WOODWARD, 1987). Entretanto, essa modificação deve ser realizada de forma restrita para não provocar o aparecimento de fluxos ilegais de informação.

5.3 Esquema de Autorização *JaCoWeb* Obrigatório

O modelo de segurança do padrão CORBA possui interfaces para utilização de políticas de controle de acesso discricionárias. Temos o objetivo de **estender as interfaces existentes no CORBAsec** usando o modelo Bell e Lapadula para incorporar **políticas obrigatórias**, inexistentes no padrão atual (OMG, 2000a).

O trabalho desenvolvido por (KARJOTH, 1998) propõe uma idéia inicial, bastante simplificada, da inclusão de políticas de controle de acesso obrigatórias no CORBAsec. O autor não aborda o uso de técnicas para evitar a superclassificação dos objetos e nem o uso de sua abordagem em um ambiente de implementação. Vários outros trabalhos relacionam o uso do modelo Bell e Lapadula com o modelo de objetos geral da programação orientada a objetos (JAJODIA e KOGAN, 1990, MILLENT e LUNT, 1992, NICOMETTE, 1997).

Esta seção irá apresentar as diferentes **entidades** que compõem o nosso modelo, a forma de **atribuir rótulos de segurança** a essas entidades, para podermos realizar um controle obrigatório.

Apresentará também as **regras** do Esquema de Autorização *JaCoWeb* Obrigatório e a forma de **estabelecer** e **usar** uma política obrigatória **usando o *PoliCap***, finalizando com um **exemplo**.

5.3.1 Entidades

As políticas obrigatórias usam rótulos de segurança para regular como os sujeitos podem acessar os objetos. Quais são os "sujeitos" e os "objetos" em um modelo de objetos ? Os **sujeitos** são os objetos enviando requisições de métodos, os **objetos** são os próprios objetos que executam os métodos. A informação flui entre os objetos na forma de **mensagens** e valores de retorno das chamadas de métodos. Esse fluxo de informações será designado como **requisições**. Quando ocorre o processo de *login*, é criado um objeto que representa o usuário e a ele é atribuída a habilitação do usuário.

Dessa forma, as entidades que fazem parte do nosso modelo são: **sujeitos, objetos** e **requisições**. Um sujeito representa uma entidade ativa que provoca a ocorrência de operações nas informações. Pode ser um um objeto que atua em favor de uma pessoa física ou os processos confiáveis do próprio ambiente. Um objeto é uma entidade passiva, é o destino das operações que podem ser executadas pelos sujeitos, e possui um estado associado. Quando um sujeito (uma entidade ativa) é o destino de uma operação (por exemplo, na comunicação entre processos), um sujeito pode também atuar como um objeto.

Para evitar a superclassificação, usaremos a técnica dos processos confiáveis em conjunto com o conceito de objetos sem estado, associando-se a esse tipo de objetos os intervalos de confiança (através dos quais o sujeito pode ler e escrever objetos de forma mais flexível do que a propriedade-* tradicional possibilitaria). Usaremos também a técnica da sensibilidade inicial assumir o valor NÃO-CLASSIFICADO e rótulos flutuantes sugeridos por (WOODWARD, 1987).

5.3.2 Os diferentes rótulos

O objetivo dos controles obrigatórios é impedir aos usuários a obtenção de informações às quais eles não estão habilitados. Supõe-se que cada operação executada no sistema é realizada por meio de uma requisição, e, dessa forma, deve-se impedir os fluxos ilegais entre usuários e impedir fluxos ilegais de informação entre diferentes requisições no sistema. Deve-se portanto, controlar todas as interações entre objetos clientes e servidores, ou, de outra forma, controlar cada requisição que acessa um objeto. De forma a realizar esse tipo de controle de acesso, deve-se **atribuir rótulos de segurança às diferentes entidades** do nosso modelo. Atribui-se um rótulo de segurança a todo sujeito e objeto do sistema (cliente ou servidor), distinguindo objetos com estado e sem estado. A fim de controlar cada interação entre objetos, deve-se rotular também cada requisição.

Cada um dos rótulos de segurança pode estar associado a um sujeito (representando sua habilitação), a um método do objeto (representado sua classificação) ou a uma requisição (representando a classificação da informação que está sendo transportada). Por comodidade, a noção de categoria não será considerada, não alterando em nada a aplicação das regras do modelo. Cada rótulo de segurança é materializado nas estruturas do modelo de controle de acesso do CORBAsec.

O modelo de controle de acesso do CORBAsec impõe naturalmente que a política obrigatória seja implementada de forma multinível, considerando cada método de uma interface como uma entidade a ser protegida e que recebe um rótulo de segurança associado.

5.3.2.1 Os sujeitos

A cada sujeito é atribuído um único rótulo de segurança f_s. Esse rótulo representa a sua **habilitação** (*clearance*) no sistema.

Para incorporar a habilitação de sujeitos na estrutura do CORBAsec, é incluída na lista de acesso representada pelo objeto ***DomainAccessPolicy*** uma coluna de dados chamada ***Clearance***[27]. O conteúdo de *Clearance* pode assumir um dos seguintes valores: NÃO-CLASSIFICADO (= 1), CONFIDENCIAL (= 2), SECRETO (= 3), ULTRA-SECRETO (= 4). Neste sentido, o objeto *DomainAccessPolicy* toma a representação da Figura 5.1, onde é associada a habilitação (ou nível de segurança) SECRETO ao sujeito *gerente_do_banco*.

Atributo de Privilégio	**Estado de Delegação**	**Direitos Efetivos (*Granted Rights*)**	***Clearance***
role:gerente_do_banco	initiator	corba: gs-- other: -u---	3

Figura 5.1 – Objeto *DomainAccessPolicy* com a inclusão da coluna *Clearance*.

5.3.2.2 Os objetos

Rótulos de segurança são atribuídos a objetos **com estado** e objetos **sem estado**.

Um **rótulo único** f_0 é associado a um objeto **com estado** *O*. Esse rótulo representa a classificação do objeto. Esse nível de segurança é fixo e não pode ser modificado durante o tempo de vida do objeto.

Um **intervalo de confiança**[28] $[fmin_0, fmax_0]$, onde $fmin_0 \leq fmax_0$, é associado com os objetos **sem estado** *O*. O rótulo $fmax_0$ representa o nível de segurança **máximo** das informações

[27] É importante salientar a diferença entre essa nova coluna introduzida na política e o atributo de privilégio *clearance*, já existente no CORBAsec. A nova coluna *clearance* introduzida no objeto de política, impõe um nível de habilitação a cada um dos atributos de privilégios, independente de ser um atributo do tipo *group*, *role*, etc.

[28] Um intervalo de confiança é definido como: $\forall x \in [fmin_0, fmax_0] \Leftrightarrow fmin_0 \leq x \leq fmax_0$.

contidas no objeto sem estado *O*. Da mesma forma, o rótulo $fmin_0$ representa o nível de segurança **mínimo** do cliente para escrever neste objeto sem estado.

Nossa proposta é representar um **intervalo de confiança** usando **dois dígitos**: o primeiro dígito representa o nível $fmin_0$ e o segundo dígito representa o nível $fmax_0$. Considerando os níveis de segurança com valores numéricos 1, 2, 3 ou 4 (NÃO-CLASSIFICADO = 1, CONFIDENCIAL = 2, SECRETO = 3, ULTRA-SECRETO = 4), podemos representar os seguintes intervalos de confiança possíveis:

Valor Associado ao Intervalo de Confiança	$fmin_0$	$fmax_0$
11	[NÃO-CLASSIFICADO,	NÃO-CLASSIFICADO]
12	[NÃO-CLASSIFICADO,	CONFIDENCIAL]
13	[NÃO-CLASSIFICADO,	SECRETO]
14	[NÃO-CLASSIFICADO,	ULTRA-SECRETO]
22	[CONFIDENCIAL,	CONFIDENCIAL]
23	[CONFIDENCIAL,	SECRETO]
24	[CONFIDENCIAL,	ULTRA-SECRETO]
33	[SECRETO,	SECRETO]
34	[SECRETO,	ULTRA-SECRETO]
44	[ULTRA-SECRETO	ULTRA-SECRETO]

Tabela 5.1 – Valores que representam os intervalos de confiança.

A utilidade dos objetos sem estado será evidenciada nas seções subseqüentes. Aqui, a título de ilustração, mostramos um exemplo que evidencia certas propriedades neste conceito.

Supondo que um objeto sem estado contém um cavalo de Tróia que conserva localmente uma cópia de todos os dados transmitidos pelas requisições antecedentes. O rótulo $fmin_0$ permite a garantia de que a cópia desses dados assim criados tem um nível de segurança superior ou igual a esse nível. Isso significa que outro objeto (leitor) deverá necessariamente possuir um nível de segurança superior ou igual a esse nível para que haja o fluxo das informações na direção deste objeto leitor.

A definição do intervalo de confiança deve considerar as informações que podem fluir através do objeto sem estado. Imaginamos, por exemplo, que o administrador de um sistema decide utilizar um novo servidor NFS distribuído livremente e gratuito de um servidor da Internet. O administrador estima a confiança que ele pode atribuir a esse servidor e assim associa ao mesmo um intervalo de confiança que não pode prejudicar as informações mais sensíveis do seu sistema. É responsabilidade do administrador do sistema definir os intervalos de confiança adequados aos objetos sem estado do sistema.

A nossa preocupação então, uma vez definida as codificações de classificações envolvendo objeto com estado e sem estado, é como representar em um ambiente CORBA estas codificações.

Considerando que as informações sobre os objetos servidores de aplicação no CORBAsec estão contidas no objeto *RequiredRights*, para associar rótulos de segurança com esses objetos, a nossa proposta inclui um novo elemento no objeto ***RequiredRights*** chamado ***Classification***. Essa extensão consiste em uma coluna que contém valores numéricos compostos por três dígitos:

- primeiro dígito: representa o **modo de acesso** do método considerado
  (1 – leitura, 2 – escrita, 3 – leitura/escrita)
- segundo e terceiro dígitos: representam a **classificação** dos objetos com estado e os intervalos de confiança associados aos objetos sem estado:
  - se o objeto é **com estado**, temos os seguintes valores: NÃO-CLASSIFICADO = 01, CONFIDENCIAL = 02, SECRETO = 03, ULTRA-SECRETO = 04
  - se o objeto é **sem estado**, os valores associados aos intervalos de confiança são os seguintes: 11, 12, 13, 14, 22, 23, 24, 33, 34 (ver Tabela 5.1).

A Figura 5.2 apresenta o objeto *RequiredRights* com a inserção da coluna *Classification*, onde a operação *Ler_valor_conta* da interface *Conta_bancária* tem como modo de acesso o valor igual a 1 (leitura), e é um objeto com estado de classificação NÃO-CLASSIFICADA (valor igual a 01). Já a operação *Imprime_saldo* da interface *Impressora_I1* é uma operação de leitura/escrita (valor igual a 3) e é um objeto sem estado com intervalo de confiança igual a 12 ([NÃO-CLASSIFICADO, CONFIDENCIAL]).

Required Rights	*Rights Combinator*	*Operation*	*Interface*	*Classification*
Corba:g---	All	Ler_valor_conta	Conta_bancária	101
Corba:-s--	All	Creditar_conta	Conta_bancária	201
Corba:-s--	All	Debitar_conta	Conta_bancária	301
Other:-s--	All	Cálculo_valor	Juros	223
Corba:gs--	All	Cálculo_rendimento	Poupança	324
Corba:gs--	All	Imprime_saldo	Impressora_I1	312

Figura 5.2 – Objeto *RequiredRights* com a inclusão da coluna *Classification*.

5.3.2.3 As requisições

Um modelo de segurança orientado a objetos deve levar em consideração os fluxos de informações que ocorrem como resultado do modelo de programação baseado em trocas de mensagens (MILLEN e LUNT, 1992).

Cada **requisição** no sistema transporta informações que podem ser modificadas pelos diferentes objetos acessados. A requisição é então, no nosso modelo, considerada uma **entidade ativa** à qual é atribuída um rótulo de segurança, caracterizando um **intervalo de confiança** representado por $[fmin_r, fmax_r]$. O elemento $fmin_r$ representa a **classificação da informação** contida na requisição e que pode ser escrita em um objeto[29]. O elemento $fmax_r$ de uma requisição representa a sua **habilitação**. Ele é inicializado com o nível corrente de segurança do principal que ativa a requisição e representa a classificação máxima das informações que podem ser lidas por esta requisição. Esses dois elementos compõem então o rótulo de segurança da requisição: **[classificação da informação, habilitação da requisição]**.

O conceito de **rótulos flutuantes** (WOODWARD, 1987) pode ser usado durante a execução de uma requisição. O rótulo associado a uma requisição, conforme o objeto servidor desta requisição, não necessariamente será o mesmo da resposta ou de uma nova requisição gerada a partir do processamento associado.

A Figura 5.3 apresenta um exemplo mostrando a flutuação de rótulos. A primeira requisição r_1 acessa um objeto para executar um dos seus métodos. Essa execução pode produzir o envio de uma nova mensagem, a requisição r_2, e essa nova requisição é rotulada levando em consideração as informações de r_1 e do *Objeto 1*. O rótulo inicial da requisição r_1 tem o valor NÃO-CLASSIFICADO (primeira ativação de requisição), e o valor SECRETO como habilitação, espelhando o usuário que invoca a operação no *Objeto 1*. O método invocado no *Objeto 1* ativa outro método a ser realizado pelo *Objeto 2*. A requisição *r2* é então rotulada como [CONFIDENCIAL, SECRETO], já que a classificação do objeto requisitante agora é CONFIDENCIAL e a requisição transporta informações sensíveis do *Objeto 1* para o *Objeto 2*.

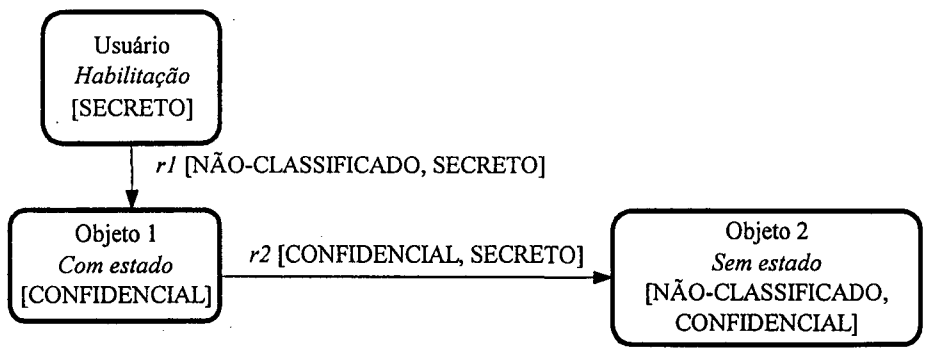


Figura 5.3 – Exemplo de mudança de rótulos das requisições.

A codificação de rótulos no modelo considerado usa a estrutura do *Request* CORBA que contém um campo de rótulo de segurança na requisição. A notação dos intervalos de confiança

[29] A classificação da primeira ativação de uma requisição criada por um usuário é inicializada com o nível de segurança = NÃO-CLASSIFICADO, já que não carrega nenhum dado sensível, seguindo a sugestão de (WOODWARD, 1987).

descrita na seção 5.3.2.2 é usada na atribuição de valores para estes rótulos. Por exemplo, o rótulo de *r1* da Figura 5.3 será representado pelo número 13 e o rótulo de r_2 ocupa o campo do *Request* com o valor 23.

A estrutura do *Request* CORBA, além de transportar campos adicionais relacionados com as *capabilities* (conforme a seção 4.3), deve transportar o rótulo de segurança da requisição tomando a forma apresentada na Figura 5.4.

Request novo com a *capability* e ***rótulo de segurança***					
Request CORBA			Campos adicionados		
IOR servidor	Método solicitado	...	Identificador Emissor	*Nonce*	**Rótulo de Segurança**

Figura 5.4 – Estrutura do *Request* com o campo do rótulo de segurança.

5.3.3 As regras do esquema de autorização obrigatório no projeto JaCoWeb

As regras do esquema de autorização definem a evolução do modelo e como são feitos os controles obrigatórios, descritos nesta seção e no capítulo 2. Para as nossas implementações são consideradas seis regras que definem a evolução dos rótulos de segurança no ambiente.

5.3.3.1 Princípios do controle de acesso obrigatório

Na apresentação das regras do esquema de autorização, serão feitas referências seguidas aos modos de acesso de leitura e escrita. Para isto, é necessário que as expressões **leitura** e **escrita** tenham um sentido bem preciso. O **acesso de leitura** sobre um objeto **com estado** O é toda execução de método de *O* que provoca um fluxo de informação de estado de *O* para a requisição de execução do método. De igual forma, o **acesso de escrita** representa toda execução de um método de *O* que provoca um fluxo de informação da requisição que executa o método para o estado do objeto *O*. O **acesso de leitura-escrita** representa toda execução de método que provoca uma troca de informações nos dois sentidos entre a requisição que executa o método e o estado do objeto *O*.

Também temos que considerar o comportamento dos dois tipos de objetos identificados no nosso modelo: objetos com estado e objetos sem estado.

Objeto com estado

Os princípios descritos para os objetos com estado derivam das duas propriedades do modelo Bell e Lapadula: a propriedade-ss e a propriedade-*.

- **Propriedade-ss (simples)**: Uma requisição de **leitura** *r* sobre um objeto com estado *O* é autorizada se e somente se $f_O \leq f\max_r$. O rótulo superior da requisição representa sua habilitação. Portanto, a requisição tem autorização para ler um objeto se sua habilitação domina a classificação do objeto.

- **Propriedade-* (estrela)**: Uma requisição de **escrita** *r* sobre um objeto com estado *O* é autorizada se e somente se $fmin_r \leq f_O$. Como a escrita é um fluxo de informação unicamente da requisição para o objeto, e, como a classificação da informação carregada pela requisição é representada pelo seu rótulo inferior, é necessário que este rótulo seja dominado pelo rótulo do objeto, para que a escrita se efetue sem fluxo de informações ilegais.

Para se garantir, então, essas propriedades, é associado a cada método de um objeto com estado um atributo que indique qual **modo de acesso** é realizado sobre o objeto na execução do método: acesso de **leitura**, de **escrita** ou de **leitura-escrita**. Esse atributo foi incluído no objeto *RequiredRights* do CORBAsec, conforme explicado na seção 5.3.2.2. Por exemplo, na Figura 5.2, a operação *Ler_valor_conta* da interface *Conta_bancária*, tem modo de acesso igual a *1* ('1'01), indicado na coluna *Classification* pelo primeiro dígito da codificação 101, que representa acesso de leitura.

Objeto sem estado

Uma requisição *r* pode acessar um objeto sem estado *O* (invocando um método qualquer de *O*) se e somente se a intersecção de $[fmin_O, fmax_O]$ e $[fmin_r, fmax_r]$ não é vazia. Isso significa que valem ambos:

- $\mathbf{fmin_O \leq fmax_r}$: determinando que a **requisição** está habilitada a **receber** informações armazenadas em *O*. É uma releitura da propriedade-ss para objetos sem estado, que também inviabiliza fluxos ilegais.
- $\mathbf{fmin_r \leq fmax_O}$: indicando que o **objeto** *O* está habilitado a **receber** informações da requisição. Isto também pode ser visto como uma releitura da propriedade-* para objetos sem estado.

Os objetos sem estado foram introduzidos em (NICOMETTE, 1996) no sentido de tratar a superclassificação da informação no Bell e Lapadula.

5.3.3.2 A flutuação dos rótulos

Neste subitem, são descritas as seis regras que lidam com o controle de autorização de uma *requisição r* que invoca um *objeto O*. As quatro primeiras regras derivam das propriedades do modelo Bell e Lapadula original, mas a quarta regra não espelha a regra original do BLP, pois leva em consideração técnicas para suavizar a superclassificação de informações. A quinta regra está relacionada com os objetos sem estado, e a regra seis trata as mensagens de retorno das invocações.

A evolução dos rótulos adaptadas ao contexto CORBAsec considera a requisição *r* representada nas estruturas de um *Request* CORBA. Os rótulos de segurança usados nas

verificações estão associados ao sujeito (rótulo presente no objeto *DomainAccessPolicy*), ao objeto (rótulo presente no objeto *RequiredRights*) e à requisição (rótulo presente no *Request* CORBA).

As informações que são geradas a partir do processamento ativado por uma *requisição r*, serão transportadas na **resposta** (retorno ao cliente) ou em uma **nova requisição**. Tanto a resposta quanto a nova requisição são representadas nas figuras subseqüentes como *requisição r'*. O rótulo associado a esta *requisição r'* é gerado em função do rótulo da *requisição r* e do *objeto O* invocado, conforme será visto a seguir na descrição das regras.

Se o objeto O é um objeto **com estado**:

- **Regra 1 (R1)** : Se o método *m* corresponde a um acesso de **leitura** e se $f_O \leq f\max_r$, o acesso **é autorizado com restrição**. O $f\min_{r'}$ que representará a execução da requisição será ajustado para o valor $\max(f\min_r, f_O)$. Essa regra traduz o fato de que o nível corrente das informações de saída do processamento é ajustado ao nível do objeto lido.

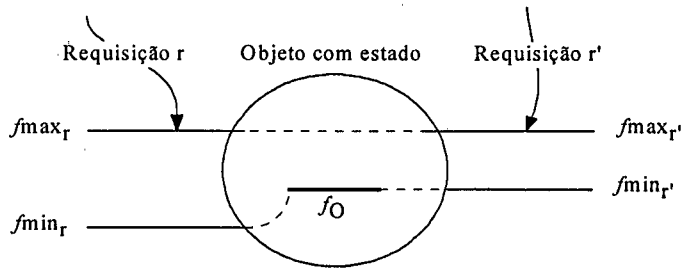


Figura 5.5 – Regra 1: Acesso de leitura com restrição a um objeto com estado.

Nota-se um caso particular dessa regra quando $f_O \leq f\min_r$. Nesse caso o acesso **é autorizado sem restrição** porque $\max(f\min_r, f_O) = f\min_r$ (Figura 5.6).

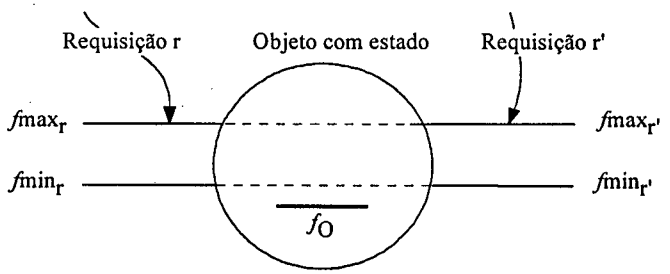


Figura 5.6 – Regra 1: Acesso de leitura sem restrição a um objeto com estado.

A regra R1 descreve a evolução das requisições que efetuam um acesso de leitura sobre um objeto com estado. Se uma *requisição r* efetua um acesso de leitura sobre um objeto, cuja classificação domina a classificação da informação veiculada pela *requisição r* (representada pelo elemento inferior do rótulo da *requisição r*), então o elemento inferior do rótulo da *requisição r'* deve ser ajustado/aumentado para a classificação do objeto (R1). Em contrapartida, uma *requisição r* não pode ler um objeto com estado cuja classificação domina a habilitação da *requisição r*.

- **Regra 2 (R2)** : Se o método *m* corresponde a um acesso de **escrita** e se $fmin_r \leq f_O$, o acesso **é autorizado sem restrição.**

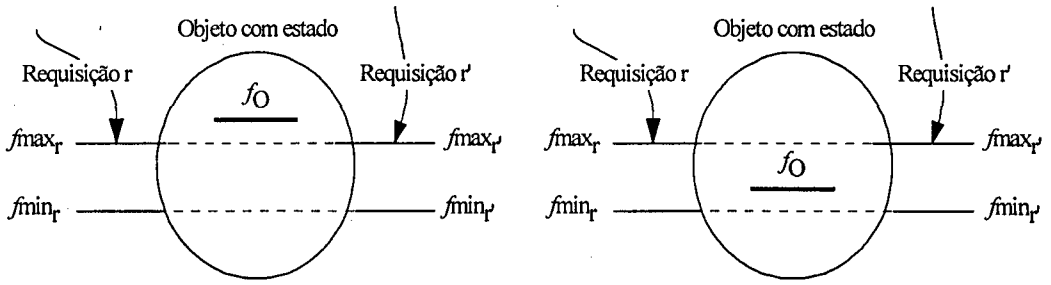


Figura 5.7 – Regra 2: Acesso de escrita sem restrição a um objeto com estado.

A regra R2 representa a propriedade estrela do modelo Bell e Lapadula quando da escrita em um objeto com estado *O*. O acesso é recusado quando a classificação do objeto é dominada pelo rótulo inferior da *requisição r*, e o acesso é concedido de forma irrestrita quando a classificação do objeto domina o elemento inferior do rótulo da *requisição r*. Em um acesso de escrita, a classificação da informação transportada pela *requisição r'* de saída **não varia**, pois um acesso de escrita consiste unicamente de um fluxo de informação da *requisição r* para o objeto.

- **Regra 3 (R3)** : Se o método *m* corresponde a um acesso de **leitura-escrita** e se f_O se situa no intervalo $[fmin_r, fmax_r]$, o acesso **é autorizado com restrição** e $fmin_{r'}$ deve ser ajustado para o valor $max(fmin_r, f_O)$. A restrição provém da operação de leitura, a escrita entretanto é autorizada sem restrição.

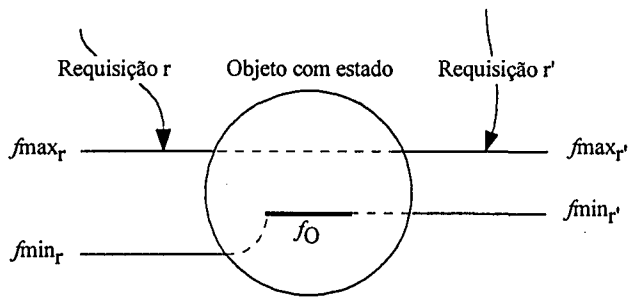


Figura 5.8 – Regra 3: Acesso de leitura-escrita com restrição a um objeto com estado.

No caso onde $f_O = fmin_r$, o acesso **é autorizado sem restrição.**

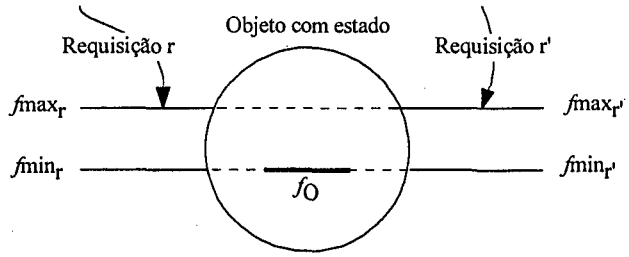


Figura 5.9 – Regra 3: Acesso de leitura-escrita sem restrição a um objeto com estado.

- **Regra 4 (R4) : Criação de objetos com estado** : Um sujeito tipicamente cria um objeto pela emissão de uma *requisição r* do tipo "Crie instância" para a classe do objeto a ser criado. A fim de evitar o aparecimento de canais cobertos, e suavizar a superclassificação da informação (WOODWARD, 1987), o **objeto "filho"** deve possuir, como rótulo inicial, um valor igual a **pelo menos a classificação da *requisição r*** que gerou essa criação (definida pelo elemento inferior do rótulo da *requisição r*). Assim, o objeto com estado é criado com $f_O = fmin_r$.

  Os objetos sem estado são criados por aplicações administrativas ou autorizadas pelo administrador de segurança do sistema. É responsabilidade deste administrador a definição dos intervalos de confiança associados aos objetos sem estado.

Se o objeto O é um objeto **sem estado:**

- **Regra 5 (R5)** : Se $fmin_O \leq fmax_r$ **e** se $fmin_r \leq fmax_O$ o acesso **é autorizado com restrição (intersecção não vazia dos intervalos de confiança).**

  Essa restrição significa que a *requisição r* só será executada se houver intersecção entre os intervalos de confiança da *requisição r* e do objeto sem estado *O*, respectivamente. As informações geradas a partir do processamento ativado serão transportadas na *requisição r'* de saída com rótulo dado por [max ($fmin_r$, $fmin_O$) , min ($fmax_r$, $fmax_O$)].

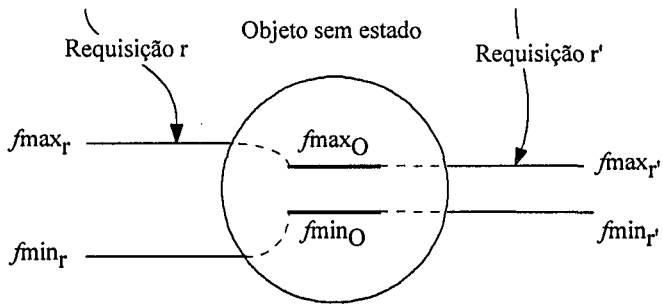


Figura 5.10 – Regra 5: Acesso com restrição a um objeto sem estado.

A regra R5, representada na Figura 5.10, descreve a evolução (ou "flutuação") do intervalo de confiança nos fluxos de requisições atravessando objetos sem estado.

Se uma ***requisição r*** de entrada acessa um objeto sem estado, cujo **elemento inferior do rótulo domina** a classificação da informação contida na *requisição r* (representada pelo elemento inferior do rótulo da *requisição r*), então o elemento inferior do rótulo da *requisição r'* deve ser elevado, assumindo o valor do elemento inferior do rótulo do objeto. Isso significa de fato que o nível corrente das informações transportadas pela *requisição r'* deve ser ajustado ao nível mínimo do objeto sem estado. *Se o* objeto sem estado tem o **elemento superior do rótulo dominado** pela habilitação da *requisição r* (representado pelo elemento superior do rótulo da *requisição r*), então a habilitação da *requisição r'* deve ser diminuída de acordo com o elemento superior do rótulo do objeto, para impedir o fluxo de informações de um outro objeto de nível superior à habilitação da *requisição r'*.

A Figura 5.11 representa o acesso negado de requisições segundo a regra R5.

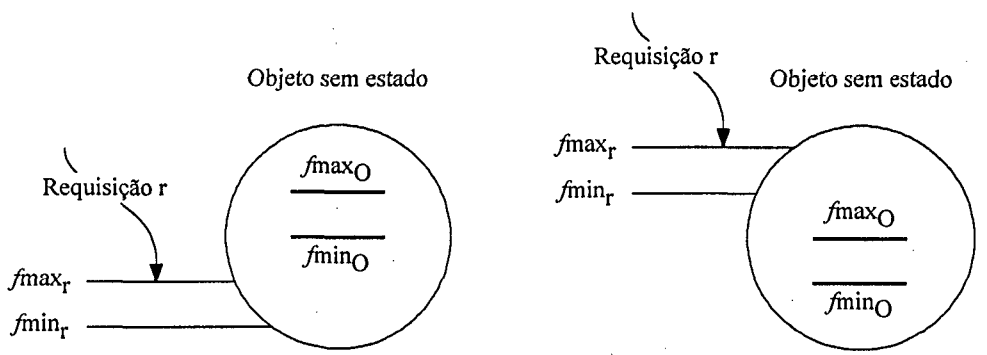


Figura 5.11 – Regra 5: Acesso recusado a um objeto sem estado.

Nota-se que a Figura 5.12 representa um caso particular da regra R5 onde o intervalo $[fmin_r, fmax_r]$ está incluso no intervalo $[fmin_O, fmax_O]$. Nesse caso, o acesso **é autorizado sem restrição** de ajuste da *requisição r'*, já que: $\max(fmin_r, fmin_O) = fmin_r$

$\min(fmax_r, fmax_O) = fmax_r$

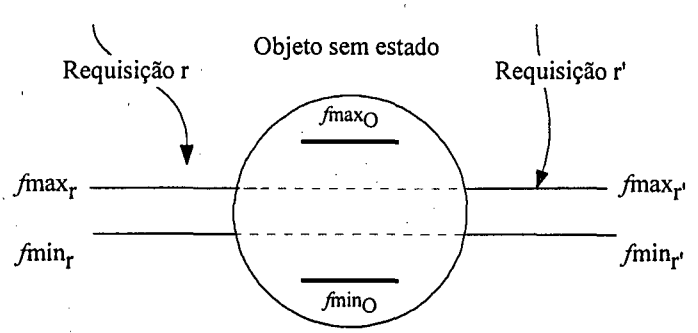


Figura 5.12 – Regra 5: Acesso sem restrição a um objeto sem estado.

Uma questão bastante importante a ser considerada nos controles obrigatórios diz respeito às mensagens de retorno das invocações (JAJODIA e KOGAN, 1990, MILLEN e LUNT, 1992). A regra 6 apresenta a nossa abordagem para tratar esta questão.

- **Regra 6 (R6) : Retorno das requisições executadas sobre objetos com estado.**

É importante notar que, em todas as regras apresentadas anteriormente, as restrições que são estabelecidas são aplicadas igualmente às mensagens de retorno ou *requisição r'*. As *requisições r'* podem ser geradas a partir de processamentos executados por objetos sem estado e por objetos com estado.

- O **objeto sem estado** não conserva informações de requisições, e, se a invocação de um método de um objeto sem estado *O* é autorizada (com ou sem restrição), o retorno dessa *requisição r* será necessariamente autorizado: o intervalo de confiança da *requisição r'* que efetua esse retorno é forçosamente incluso no intervalo de confiança do objeto chamador.
- No caso dos **objetos com estado**, uma *requisição r*, invocando uma operação de **leitura** ou de **leitura-escrita**, sobre um objeto *O*, pode causar uma *requisição r'*. Esta *requisição r'* representa a mensagem de retorno e causa uma **escrita** no objeto chamador. Nesse caso, é necessário controlar essa escrita da mesma forma que a chamada de um método de escrita de um objeto qualquer. A *requisição r'* tem acesso de escrita sem restrição ao objeto chamador caso cumpra uma das seguintes condições:
  - Se o objeto chamador for um objeto com estado: $fmin_{r'} \leq f_{\text{OBJETO CHAMADOR}}$
  - Se o objeto chamador for um objeto sem estado: $fmin_{r'} \leq fmax_{\text{OBJETO CHAMADOR}}$

  Caso nenhuma dessas condições seja satisfeita pela *requisição r'*, o acesso de escrita é negado.

5.3.4 Restrições com RBAC

No modelo RBAC[30] existe o conceito de restrições (*constraints*) estabelecidas sobre os papéis mutuamente exclusivos. Considerando o *funcionário contábil* e o *gerente contábil* que participam na emissão de um cheque de uma empresa, verificamos que esses dois papéis da empresa são, necessariamente, mutuamente exclusivos. Isso porque não podem ser assumidos pela mesma pessoa, já que isso aumentaria a probabilidade de fraudes.

No trabalho desenvolvido por Konstantin (BEZNOSOV e DENG, 1999), os autores definem um *framework* para prover a implementação de RBAC usando o CORBAsec. Todos os componentes deste *framework* são definidos de forma bastante geral, e funcionam como um guia para uma possível implementação. Neste trabalho, é definido que as restrições na ativação de

[30] Ver capítulo 2.

papéis devem ser implementadas pelo objeto *PrincipalAuthenticator* no momento da autenticação no CORBAsec.

Nosso objetivo no esquema de autorização *JaCoWeb* obrigatório é estabelecer **restrições nos papéis assumidos pelos principais no modelo**, de forma a evitar a ocorrência simultânea para um principal de papéis mutuamente exclusivos. No nosso modelo, o estabelecimento de restrições também deve ser implementado pelo objeto *PrincipalAuthenticator*. Na aplicação implementada no protótipo (seção 4.4), ficou evidente a necessidade da existência deste requisito. Por exemplo, os papéis **cliente** e **gerente** do banco são mutuamente exclusivos. Isso porque, em uma mesma sessão, caso esses dois papéis sejam ativados simultaneamente pelo mesmo principal, podem aparecer fluxos ilegais de informações, visto que a habilitação associada a cada um destes papéis é certamente diferente.

Os controles de papéis do nosso modelo está baseado em uma **separação dinâmica** de papéis. Na separação dinâmica, a exclusão mútua entre dois papéis *R1* e *R2* significa que um usuário pode ser associado a ambos, desde que apenas um deles (*R1* ou *R2*) esteja ativo em um dado momento (SANDHU, FERRAIOLO, *et al.*, 2000). Um estudo mais detalhado e completo sobre separação dinâmica de papéis está sendo feito em (OBELHEIRO, 2000).

5.3.5 Estabelecimento e Uso da Política Obrigatória usando o PoliCap

Aplicações administrativas, executadas pelo administrador do sistema, cooperam para o estabelecimento da política obrigatória a ser armazenada no *PoliCap*. Como os objetos *DomainAccessPolicy* e *RequiredRights* são modificados, acrescentado-se aos mesmos as colunas *Clearance* e *Classification*, respectivamente, a IDL desses dois objetos deve ser modificada para comportar os controles obrigatórios correspondentes. A IDL do *PoliCap* permanece igual.

Para o objeto ***DomainAccessPolicy***, definimos os seguintes métodos:

- ***grant_clearance***: este método insere o nível de habilitação do sujeito;
- ***revoke_clearance***: este método remove o nível de habilitação do sujeito;
- ***replace_clearance***: este método substitui a habilitação do sujeito;
- ***get_clearance***: este método consulta a habilitação de um sujeito.

Dessa forma, a IDL do objeto *DomainAccessPolicy* é acrescida com as seguintes definições:

```
interface DomainAccessPolicy : AccessPolicy {
    ...
    void grant_clearance(
        in   Security::SecAttribute      priv_attr,
        in   Security::DelegationState   del_state,
        in   Security::ExtensibleFamily  rights_family,
        in   Security::SecAttribute      clearance
    );
```

```
        void revoke_clearance(
            in   Security::SecAttribute      priv_attr,
            in   Security::DelegationState   del_state,
            in   Security::ExtensibleFamily  rights_family,
            in   Security::SecAttribute      clearance
        );

        void replace_clearance(
            in   Security::SecAttribute      priv_attr,
            in   Security::DelegationState   del_state,
            in   Security::ExtensibleFamily  rights_family,
            in   Security:: SecAttribute     clearance
        );

        Security::SecAttribute get_clearance (
            in   Security::SecAttribute      priv_attr,
            in   Security::DelegationState   del_state,
            in   Security::ExtensibleFamily  rights_family
        );
    };
```

Para o objeto ***RequiredRights***, definimos os seguintes métodos:

- ***set_classification***: este método insere a classificação do método do objeto;
- ***get_classification***: este método consulta a classificação do método do objeto.

Dessa forma, a IDL do objeto *RequiredRights* é acrescida com as seguintes definições:

```
interface RequiredRights{
    ...
    void get_classification (
        in   Object                          obj,
        in   CORBA::Identifier               operation_name,
        in   CORBA::RepositoryId             interface_name,
        out  Security::SecAttribute          classfication
    );

    void set_classification (
        in   string                          operation_name,
        in   CORBA::RepositoryId             interface_name,
        in   Security:: SecAttribute         classification
    );
};
```

Assim, o estabelecimento da política obrigatória segue procedimentos semelhantes aos descritos no estabelecimento da política discricionária (seção 4.2.1), visto que a única diferença entre os dois tipos de políticas, considerando as estruturas de dados do CORBAsec, é o acréscimo de colunas nos objetos *DomainAccessPolicy* e *RequiredRights*.

A única parte que executa funções especiais, no caso das políticas obrigatórias, é o administrador de segurança do sistema. O **próprio administrador** do ambiente, executando uma aplicação administrativa, além das ações referidas à implantação de políticas discricionárias, realiza as seguintes tarefas:

- define a **habilitação** dos sujeitos usando os métodos *grant_clearance*, *revoke_clearance*, *replace_clearance* e *get_clearance*, propostos no objeto *DomainAccessPolicy*;

- especifica a **classificação** das operações existentes na interface de cada um dos objetos, usando a operação *set_classification*, proposta no objeto *RequiredRights*.

A evolução dos rótulos da política obrigatória acontece conforme ocorre o processo de ligação entre objetos cliente e servidor. Em ***bind time*** (*tempo de ligação*) a política obrigatória, juntamente com a discricionária, é verificada de forma global, através da interceptação no lado do cliente, a nível de controle de acesso, desviando para o *PoliCap*. Uma vez verificada esta política, são montados os objetos locais no lado do cliente. Ao montar os objetos locais, a política obrigatória passa a ser representada nos objetos *DomainAccessPolicy* e *RequiredRights* locais, através das colunas *Clearance* e *Classification*.

Em ***access decision time*** (*tempo de decisão de acesso*), localmente **no lado do objeto cliente**, o objeto *AccessDecision* realiza o controle obrigatório, implementando as regras R1, R2, R3, R4, R5 e R6, apresentadas na seção 5.3.3.2. Para implementar cada uma das regras, são usadas as operações para consulta dos rótulos de segurança, propostas para os objetos *DomainAccessPolicy* (*get_clearance*) e *RequiredRights* (*get_classification*). Depois da execução da regra obrigatória apropriada, o controle discricionário da invocação é executado, no lado do cliente, gerando uma *capability* (seção 4.3)[31]. Além dos valores associados a esta *capability*, o *Request* CORBA deve transportar mais um **campo**, cujo valor é o **rótulo de segurança da requisição** (seu intervalo de confiança). O objeto *AccessDecision*, no lado do servidor, faz a verificação da *capability* relativa ao controle discricionário, e extrai o rótulo de segurança correspondente ao intervalo de confiança da requisição nas verificações obrigatórias do lado do servidor.

Os rótulos de segurança no lado do servidor são também necessários para os controles de retorno ou mesmo de novas requisições, gerados a partir do servidor. Tanto as mensagens de requisição do cliente quanto as mensagens de retorno (*replies*) são rotuladas, visto que esses rótulos podem ser modificados pelas regras R1, R3, R4 e R5.

O ajuste de rótulos da *requisição r'* (mensagens de retorno ou novas requisições), na nossa proposta, é realizado ainda no lado do cliente pelo objeto *AccessDecision,* antes que ocorra o fluxo de informações da *requisição r* do lado do cliente (objeto chamador) para o lado do servidor. Dessa forma, a verificação das condições impostas pela regra R6 trata as mensagens de retorno já no lado do cliente, negando a execução de uma requisição feita por um objeto chamador com estado, que pretende realizar o acesso de leitura ou de leitura-escrita, caso este objeto chamador não cumpra as condições impostas por esta regra R6 (acesso de leitura: $f\min_{r'} \leq f_{OBJETOCHAMADOR}$; acesso de leitura-escrita: $f\min_{r'} \leq f\max_{OBJETOCHAMADOR}$). Esta verificação de condições da regra R6, feita ainda no lado

[31] Caso algum dos controles realizados (obrigatório ou discricionário), produza a negação do acesso à execução de requisições, uma mensagem de exceção é enviada para o usuário.

do cliente, simplifica os controles obrigatórios associados às mensagens de retorno e prevê somente o fluxo de requisições autorizadas no ambiente.

5.3.6 Exemplo de uso das regras do Esquema de Autorização Obrigatório

Um exemplo de uso do esquema de autorização obrigatório está sendo desenvolvido no quadro do projeto *JaCoWeb*, e é mostrado aqui no sentido de ilustrar as potencialidades dos controles realizados neste esquema. O **exemplo de aplicação bancária**, representada em *UML* (*Unified Modeling Language* (ERIKSSON e PENKER, 1998)) na Figura 4.9 do capítulo 4 foi o adotado nas demonstrações de eficiência do esquema obrigatório.

Um servidor bancário serve clientes do banco e gerentes do banco. Tanto clientes quanto gerentes gostariam de armazenar seus dados de forma segura de modo que outros usuários e não-usuários em geral não tivessem acesso a esses dados. Além disso, o próprio banco armazena informações privadas, como dados dos seus investimentos, que não devem ser vistas por usuários e não-usuários.

O banco recebe requisições periódicas de cada cliente para verificar o extrato, verificar o saldo, e fazer transferência de dinheiro entre contas. Cada requisição deveria ser capaz de observar apenas as informações que são de propriedade do cliente, e nenhuma informação privada do banco. Para atingir tal objetivo, requisições dos clientes e gerentes ficam sujeitos à **política obrigatória do esquema de autorização JaCoWeb**, aplicada sobre as interações entre os clientes/gerentes e o servidor bancário.

Para descrever o funcionamento do exemplo e os controles obrigatórios associados, será considerado um **objeto cliente** do servidor bancário, que **quer realizar a transferência de dinheiro entre contas da mesma agência e obter um comprovante impresso da operação realizada**. Os objetos envolvidos nessa operação são: um *applet* cliente, o servidor bancário, arquivos de conta e o servidor de impressão.

O diagrama de seqüência (*UML sequence*) da Figura 5.13 mostra a seqüência na execução das operações do exemplo. A figura 5.14 esclarece os níveis de classificação usados no exemplo.

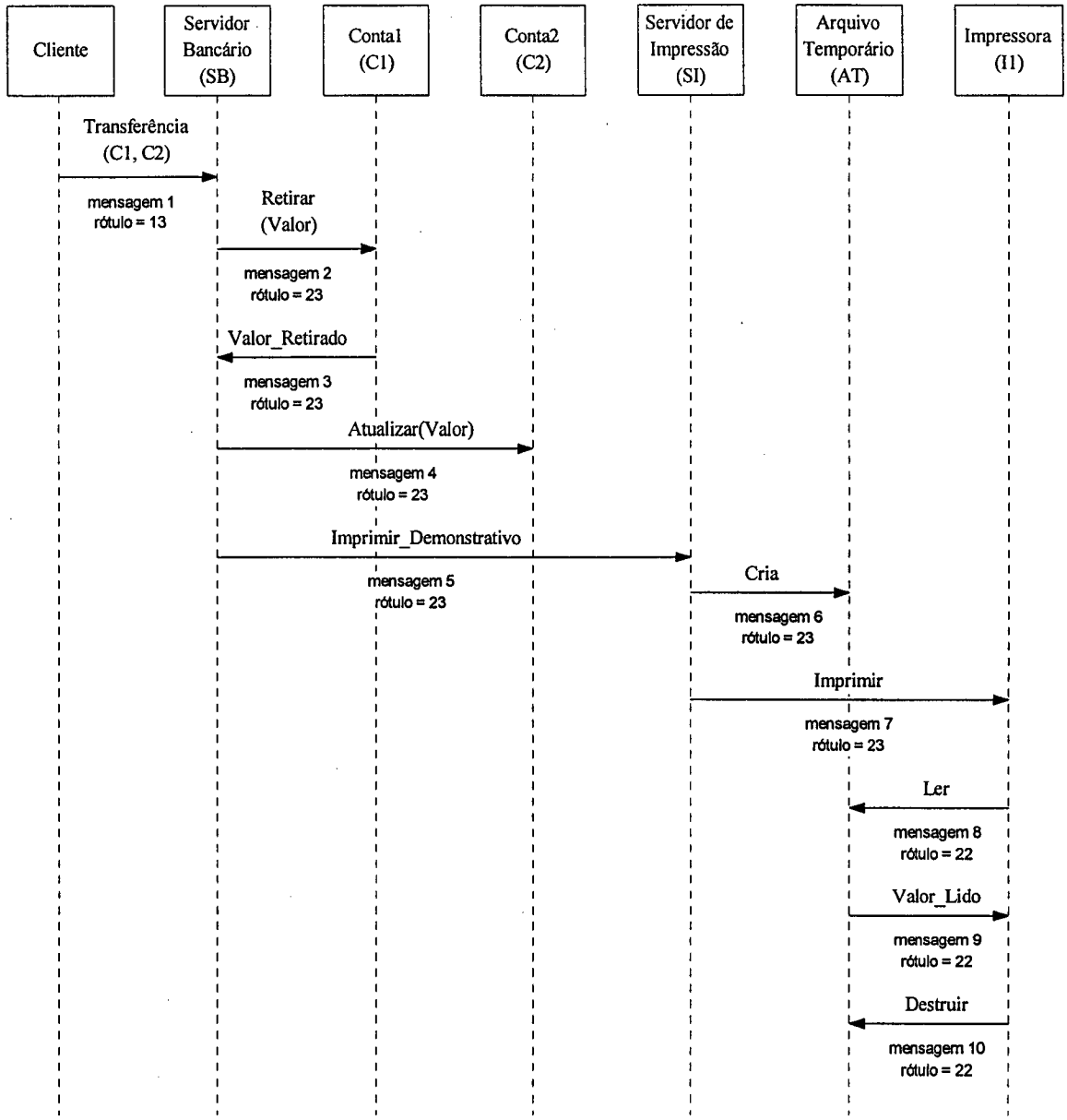


Figura 5.13 – Diagrama de Seqüência da Transferência entre contas.

Hierarquia de níveis de sensibilidade na aplicação bancária:

NÃO-CLASSIFICADO < CONFIDENCIAL < SECRETO < ULTRA-SECRETO
1 2 3 4

Rótulos dos objetos:

Cliente C1:	3 [SECRETO]
Servidor Bancário SB:	23 [CONFIDENCIAL,SECRETO] – objeto sem estado
Servidor de Impressão SI:	13 [NÃO-CLASSIFICADO,SECRETO] – objeto sem estado
Impressora I1:	12 [NÃO-CLASSIFICADO,CONFIDENCIAL] – objeto sem estado
Arquivo de Conta C1:	2 [CONFIDENCIAL] – objeto com estado
Arquivo de Conta C2:	2 [CONFIDENCIAL] – objeto com estado

Figura 5.14 – Níveis de classificação usados no exemplo.

O cenário que representa a execução da transferência entre contas e os controles obrigatórios associados é descrito a seguir. Os controles discricionários associados à execução desse exemplo pode ser entendido no contexto do serviço de política *PoliCap* do modelo *JaCoWeb*, e segue o modelo discricionário do CORBAsec.

1) O usuário *Cliente* se conecta com o sistema com habilitação SECRETO representado pelo valor numérico 3. Uma *requisição r* é disparada para executar a operação de *Transferência* de dinheiro entre contas do Servidor Bancário. Essa requisição *r* é rotulada com o valor 13 [NÃO-CLASSIFICADO, SECRETO] (**mensagem 1**). O elemento inferior do rótulo é NÃO-CLASSIFICADO porque a requisição *r* não contém ainda nenhuma informação sensível. O elemento superior do rótulo é inicializado com a habilitação do usuário que ativou a requisição *r*.

2) O *Servidor Bancário* (SB) é um objeto sem estado cujo intervalo de confiança é igual a 23 [CONFIDENCIAL, SECRETO]. A intersecção desse intervalo de confiança e do intervalo de confiança da requisição *r* é não vazia. Mas como $fmin_r \leq fmin_{SB} \leq fmax_r \leq fmax_{SB}$, o acesso é autorizado com restrição: $fmin_r$ **deve ser ajustado** para CONFIDENCIAL (Regra R5);

3) O *Servidor Bancário* (SB) invoca o método *Retirar* do objeto arquivo de contas *Conta1* (C1) (**mensagem 2**). Essa requisição *r* é rotulada com o valor 23 [CONFIDENCIAL, SECRETO]. O arquivo *Conta1* é um objeto com estado cuja classificação é CONFIDENCIAL e o acesso é de leitura-escrita. Como $f_{C1} = fmin_r$, o acesso é autorizado sem restrição (caso particular da regra R3).

4) O retorno da execução do método *Retirar* é rotulado com o valor 23 [CONFIDENCIAL, SECRETO] (**mensagem 3**), já que não houve nenhum ajuste dos rótulos na mensagem 2. Esse retorno é autorizado sem restrição já que seu intervalo de confiança é igual ao intervalo de confiança de SB (Regra R6).

5) O SB invoca o método *Atualizar* do arquivo de contas *Conta2* (**mensagem 4**). Essa requisição *r* é rotulada com o valor 23 [CONFIDENCIAL, SECRETO]. O arquivo *Conta2* (C2) é um objeto com estado cuja classificação é CONFIDENCIAL e o acesso é o acesso de escrita. Como $f\min_r \leq f_{C2}$ o acesso é autorizado sem restrição (Regra R2).

6) O SB invoca o método *Imprimir_Demonstrativo* do *Servidor de Impressão* (SI) (**mensagem 5**). Essa requisição *r* é rotulada com valor 23 [CONFIDENCIAL, SECRETO]. O *Servidor de Impressão* é um objeto sem estado cujo intervalo de confiança possui o valor 13 [NÃO-CLASSIFICADO, SECRETO]. A intersecção desse intervalo de confiança e do intervalo de confiança da requisição *r* é não vazia. Mas como $[f\min_r, f\max_r]$ está incluso no intervalo $[f\min_{SI}, f\max_{SI}]$, o acesso é autorizado sem restrição (caso particular da regra R5).

7) O SI cria um objeto *Arquivo Temporário* AT (**mensagem 6**). Essa requisição *r* de criação possui o intervalo de confiança com valor 23 [CONFIDENCIAL, SECRETO]. O arquivo AT é um objeto com estado e é criado com um rótulo que é no mínimo $f_{AT} = f\min_r$ (Regra R4). Essa criação é acompanhada da escrita de AT (os dados escritos podem ser passados como parâmetro na criação do objeto).

8) O SI invoca o método *Imprimir* da *Impressora* I1 (**mensagem 7**). Essa requisição *r* é rotulada com valor 23 [CONFIDENCIAL, SECRETO]. A impressora I1 é um objeto sem estado cujo intervalo de confiança possui o valor 12 [NÃO-CLASSIFICADO, CONFIDENCIAL]. A intersecção desse intervalo de confiança e do intervalo de confiança da requisição *r* é não vazia. Mas como $f\min_{I1} \leq f\max_r$ e $f\min_r \leq f\max_{I1}$, o acesso é autorizado com restrição: $f\max_r$ **deve ser ajustado** para CONFIDENCIAL (Regra R5).

9) A impressora I1 invoca o método *Ler* do arquivo AT (**mensagem 8**). Essa requisição *r* é rotulada com o valor 22 [CONFIDENCIAL, CONFIDENCIAL]. O objeto AT é um objeto com estado cuja classificação é CONFIDENCIAL e o acesso é de leitura. Como $f_{AT} \leq f\min_r$, o acesso é autorizado sem restrição (caso particular da regra R1).

10) O retorno dessa requisição (**mensagem 9**) é rotulado com o valor 22 [CONFIDENCIAL, CONFIDENCIAL] e é autorizado pois seu intervalo de confiança está incluso no intervalo de confiança de I1 (Regra R6).

11) I1 invoca o método *Destruir* do arquivo AT (**mensagem 10**). Essa requisição *r* é rotulada com o valor 22 [CONFIDENCIAL, CONFIDENCIAL]. O objeto AT é um objeto com estado cuja classificação é CONFIDENCIAL e o acesso é de escrita. Como $fmin_r \leq f_{AT}$, o acesso é autorizado sem restrição (regra R2).

No caso do exemplo, pode-se constatar o interesse pelos objetos sem estado e pelas outras técnicas para evitar a superclassificação, para obter uma política de segurança menos restritiva do que o Bell e Lapadula. Na verdade, o rótulo do arquivo temporário AT é atribuído em função do rótulo mínimo da requisição responsável por sua criação. Assim, o objeto SI de habilitação SECRETO cria um objeto CONFIDENCIAL pois essa criação é realizada por uma requisição rotulada [CONFIDENCIAL, SECRETO].

No caso da política Bell e Lapadula pura, sem objetos de confiança, o rótulo do objeto AT deve ser necessariamente superior ou igual ao SI. Assim, se SI é SECRETO (um objeto possui um só nível na política Bell e Lapadula), o arquito AT é criado SECRETO. A impressora I1 não poderia acessar o arquivo. Seria possível atribuir um rótulo mais baixo a SI mas então esse servidor não poderia acessar os arquivos com rótulo elevado. Logo, a operação de criação de objetos seria um problema quando adaptado ao modelo Bell e Lapadula puro, sem objetos que não tenham estados persistentes.

Para poder efetuar as mesmas operações do modelo proposto, seria necessário, no caso da política Bell e Lapadula, criar várias instâncias do objeto SI cada uma delas com um rótulo diferente. Cada instância seria invocada em função da classificação do arquivo lido, por exemplo. Da mesma forma, seria necessário várias instâncias da impressora I1, cada uma com uma etiqueta diferente. Pode-se perceber que a administração de um sistema desse tipo pode tornar-se inviável.

Com este pequeno exemplo, acreditamos ter mostrado a viabilidade do nosso controle obrigatório que permite garantir as mesmas propriedades do Bell e Lapadula, porém, de forma menos restritiva.

5.4 Trabalhos Correlatos

O trabalho desenvolvido por (KARJOTH, 1998) propõe uma idéia inicial, bastante simplificada, da inclusão de políticas de controle de acesso obrigatórias no CORBAsec. Nessa proposta, o autor atribui a habilitação aos sujeitos no objeto *DomainAccessPolicy* e a classificação aos objetos no objeto *RequiredRights*. Designa três níveis de leitura (*r1*, *r2* e *r3*) e três níveis de escrita (*w1*, *w2* e *w3*), passíveis de serem atribuídos aos métodos de cada interface. O autor não

aborda o uso de técnicas para evitar a superclassificação dos objetos e não concretiza o uso de sua abordagem em um ambiente de implementação.

O trabalho de (NICOMETTE, 1996) apresenta um esquema de autorização obrigatório, usando, para evitar a superclassificação de informações, as técnicas dos objetos sem estado e dos intervalos de confiança associados aos processos confiáveis. Este trabalho usa uma estrutura de programação orientada a objetos genérica, e define oito regras no seu esquema de autorização obrigatório. Sua abordagem não possui regras para a criação dos objetos e para tratar as mensagens de retorno em um esquema obrigatório.

Outros trabalhos relacionam o uso do modelo Bell e Lapadula com o modelo de objetos geral da programação orientada a objetos (JAJODIA e KOGAN, 1990, MILLENT e LUNT, 1992, NICOMETTE, 1997). O trabalho de (JAJODIA e KOGAN, 1990) define um algoritmo de "filtro" de mensagens, onde cada requisição tem o seu nível de "*clearance*" verificado junto ao rótulo de segurança associado ao objeto servidor. Sua abordagem considera rótulos de segurança associados aos objetos como um todo, e não a cada um dos seus métodos. A proposta de (MILLEN e LUNT, 1992) considera um ambiente orientado a objetos construído sobre um sistema operacional obrigatório tradicional.

O uso de políticas obrigatórias baseadas no modelo Bell e Lapadula também vem sendo pesquisada por (OSBORN, SANDHU, *et al.*, 2000). No seu trabalho, os autores propõem uma forma de implementar modelos obrigatórios com RBAC.

5.5 Considerações Finais

A proposta de política de autorização obrigatória para o modelo de segurança CORBA vem preencher uma lacuna nessa plataforma, concretizando uma política de segurança baseada no Bell e Lapadula.

Comparando o trabalho proposto com a literatura existente, podemos verificar que o Esquema de Autorização *JaCoWeb* Obrigatório trata as políticas obrigatórias no ambiente CORBAsec, a exemplo de (KARJOTH, 1998), mas sugere e usa técnicas para evitar a superclassificação das informações inerente ao modelo Bell e Lapadula. Não temos conhecimento de outros trabalhos da literatura que tratam as políticas obrigatórias no CORBAsec. Nesse sentido nossa proposta é inovadora.

O trabalho de (OSBORN, SANDHU, *et al.*, 2000) trata a inclusão de políticas obrigatórias em ambientes que usam o RBAC de forma bastante superficial, sem considerar ambientes práticos.

Nosso trabalho, a exemplo de (NICOMETTE, 1996), usa os objetos sem estado e os intervalos de confiança associados com as requisições para evitar a superclassificação. Entretanto,

nossa proposta de controles obrigatórios está inserida em um ambiente de objetos distribuídos acessados a partir de *browsers* Web. A nossa solução se utiliza de interceptadores do CORBAsec que permitem uma verificação transparente e também flexível. Uma das diferenças da nossa proposta, em contraste com outros trabalhos da literatura mencionada (ver Tabela 5.2), é a **concretização dos controles obrigatórios** realizados **no lado do cliente**, antes que a requisição inicie a comunicação com o objeto servidor. Isto possibilitou a existência de uma definição clara da regra que trata as mensagens de retorno (Regra R6), questão esta que normalmente é tratada de maneira informal, o que pode ser constatado em (MILLEN e LUNT, 1992, NICOMETTE, 1996). A implementação desta regra no nosso esquema não provoca bloqueios de retornos de requisições já processadas, como pode ocorrer em (JAJODIA e KOGAN, 1990, MILLEN e LUNT, 1992, NICOMETTE, 1996). Nosso trabalho define seis regras que mostram a maneira como os rótulos de segurança evoluem no sistema, usando técnicas para evitar a superclassificação de informações e levando em consideração um possível ambiente de implementação.

Característica / Trabalho	Rótulos em cada método	Usa CORBAsec	Bloqueio no retorno de requisições já processadas	Verifica no lado	Implementação	Evita superclassificação
Karjoth 1998	Sim	Sim	Não trata	Não trata	Não considera	Não
Millen e Lunt 1992	Não	Não	Pode causar	Servidor	Considera	Sim
Jajodia e Kogan 1990	Não	Não	Pode causar	Não trata	Não considera	Não
Nicomette 1996	Sim	Não	Pode causar	Servidor	Considera	Sim
JaCoWeb Obrigatório	**Sim**	**Sim**	**Não causa**	**Cliente**	**Considera**	**Sim**

Tabela 5.2 – *JaCoWeb* Obrigatório x Trabalhos Correlatos.

5.6 Conclusões do Capítulo

Este capítulo apresentou o Esquema de Autorização *JaCoWeb* Obrigatório, definindo as entidades que fazem parte do ambiente, os rótulos de segurança associados, as regras do esquema de autorização, o estabelecimento e uso de políticas obrigatórias a partir do *PoliCap* e o uso do esquema definido através de um exemplo.

O Esquema de Autorização *JaCoWeb* Obrigatório possibilita a modelagem e a concretização da política de segurança obrigatória em ambientes que usam o modelo CORBAsec. Neste sentido, o esquema proposto contribui de forma significativa na implantação de políticas de segurança obrigatórias em ambientes abertos distribuídos.

A existência dos controles obrigatórios, sobre os quais os usuários não têm controle, auxiliam a controlar as violações de segurança existentes em aplicações comerciais complexas que executam em larga escala neste ambiente, conforme afirmam (ZHONG e EDWARDS, 1998). As

políticas obrigatórias permitem estabelecer regras incontornáveis que asseguram a proteção eficaz contra a descoberta ou fuga de informações.

Nosso esquema de autorização aproveita as boas características do *PoliCap*, para estabelecer políticas discricionárias relacionadas com as políticas obrigatórias do ambiente. Para implementar as políticas obrigatórias, não foi necessária a inclusão de numerosas estruturas de dados no ambiente. Tivemos simplesmente a inclusão de um campo no objeto *DomainAccessPolicy* (habilitação do sujeito) e uma coluna no objeto *RequiredRights* (classificação do método do objeto).

No âmbito do projeto *JaCoWeb Security*, está sendo implementada uma aplicação no sentido de evidenciar as políticas de controle de acesso obrigatórias definidas em nossa proposta.

Capítulo 6

AVALIAÇÃO DO ESQUEMA DE AUTORIZAÇÃO *JACOWEB* USANDO OS CRITÉRIOS COMUNS DE SEGURANÇA

Este capítulo apresenta uma avaliação qualitativa do esquema de autorização *JaCoWeb*, considerando políticas discricionárias e obrigatórias propostas neste esquema, de acordo com os Critérios Comuns de Segurança (padrão ISO 15408) (CEMEB, 1997, CEMEB, 1999, ISO/IEC 15408-1, 1999, ISO/IEC 15408-2, 1999, ISO/IEC 15408-3, 1999).

Os Critérios Comuns de Segurança estabelecem uma metodologia para definir requisitos de segurança de um produto, para avaliar e atribuir uma classificação à segurança de produtos, e para gerar toda a documentação que prove a aplicação desses critérios.

Neste capítulo, são apresentados os conceitos básicos relativos aos Critérios Comuns e a forma de avaliar produtos usando esses critérios. O *JaCoWeb-ST*, que representa a definição dos requisitos de segurança do *Esquema de Autorização JaCoWeb*, na ótica desse padrão, é apresentado de forma resumida. Por fim, um relatório final da avaliação do *Esquema de Autorização JaCoWeb* é descrito, apresentando os resultados obtidos.

6.1 Os *Critérios Comuns* para Avaliação da Segurança

6.1.1 Visão Geral

Em geral, em razão da complexidade de sistemas informáticos e das limitações das técnicas de verificação, a segurança nesses sistemas não pode ser provada ou estimada. Entretanto, usuários de sistemas seguros necessitam algum tipo de garantia de que os produtos que eles usam fornecem segurança adequada. Esses usuários poderiam:

- Confiar na palavra do fornecedor do produto/serviço;
- Testar o sistema pessoalmente;
- Confiar em alguma declaração imparcial feita por uma entidade independente (avaliação).

Normalmente, algum tipo de avaliação da segurança é a única alternativa para se obter a confiança na segurança de um produto (GOLLMANN, 1999).

Os **princípios universais de avaliação**, definidos em (CEMEB, 1997, BELVIN, 2000) são os seguintes:

- **Adequação** (*appropriateness*): as atividades de avaliação empregadas para atingir um nível de garantia pretendido **devem ser apropriadas.**
- **Imparcialidade** (*impartiality*): todas as avaliações **devem ser livres de tendências.**
- **Objetividade** (*objectivity*): resultados de avaliação **devem ser obtidos** com um mínimo de julgamentos ou opiniões.
- **Possibilidade de repetição** (*repeatability*) e **reprodução** (*reproducibility*): a avaliação repetida dos mesmos requisitos e com as mesmas evidências de avaliação **devem fornecer os mesmos resultados.**
- **Solidez** (*soundness*) **dos resultados**: os resultados da avaliação **devem ser completos** e tecnicamente corretos.

Esses princípios de avaliação levam em consideração suposições sobre o ambiente de avaliação como (CEMEB, 1997): a relação custo/benefício do processo de avaliação, a metodologia de avaliação, a possibilidade de reutilizar resultados de avaliações anteriores, e, ainda, a existência de uma nomenclatura comum usada entre todas as partes envolvidas no processo de avaliação.

Um Critério de Segurança para tecnologias de informações corresponde a um conjunto de regras, conceitos e metodologias, cuja finalidade é permitir realizar tarefas como: **definir** os requisitos de segurança para tecnologias de informação tanto a nível de *hardware* como de *software* (ponto de vista de clientes), **descrever** as características de seus produtos específicos (ponto de vista de desenvolvedores), e **medir a confiança** a ser conferida à segurança de um produto (ponto de vista de avaliadores) (CAPLAN e SANDERS, 1999, TROY, 1999).

Os Critérios Comuns (*Common Criteria – CC*) para avaliação da segurança correspondem ao resultado de um esforço de várias organizações internacionais para desenvolver um único padrão de avaliação da segurança para sistemas e produtos de tecnologias de informação, considerando os desafios na segurança de sistemas presentes nos anos 90. Esse critério deriva de padrões anteriores como o TCSEC (ou livro laranja que data dos anos 80), o ITSEC (critério europeu de 1991), o CTCPEC (critério canadense de 1993) e o FC (união dos critérios europeu e canadense de 1993). As organizações colaboradoras que participam destes esforços são: CSE (Canadá), SCSSI (França), BSI (Alemanha), NLNCSA (Holanda), CESG (Inglaterra) e NIST e NSA (Estados Unidos) (ISO/IEC 15408-1, 1999). Estes critérios comuns de avaliação tornaram-se o padrão ISO (*International Standard Organization*) 15408 em 1999 (ISO/IEC 15408-1, 1999, ISO/IEC 15408-2, 1999, ISO/IEC 15408-3, 1999).

Os Critérios Comuns de avaliação estão fundamentados em uma *linguagem* e uma *estrutura* comuns para expressar requisitos de segurança de sistemas/produtos de tecnologia de informação e fornecem *catálogos*[32] de componentes e *pacotes*[33] de requisitos de segurança padronizados.

Estes padrões de avaliação são usados no *desenvolvimento* de *Perfis de Proteção* (*PP - Protection Profiles*) e *Alvos de Segurança* (*ST - Security Targets*), que estabelecem *o que um produto/sistema deve fazer*, através de definições de requisitos de segurança de tecnologias de informação específicos para produtos e/ou sistemas. Estes padrões também são considerados fundamento técnico completo e confiável para *avaliar* as características de segurança de produtos e sistemas, de forma a se obter a confiança de que as suas funções de segurança estão bem construídas e cumprem seus objetivos.

O CC é divido em três partes:

- **Parte 1 - Introdução e Modelo Geral**

  A parte 1 introduz o CC. Ela define conceitos gerais e princípios de avaliação de segurança de tecnologias de informação, e apresenta um modelo de avaliação geral. Esta primeira parte também define construções para expressar os objetivos de segurança de uma tecnologia de informação, para selecionar e definir requisitos de segurança da mesma, e para escrever especificações de alto nível de produtos e sistemas. Estas construções são chamadas *Perfis de Proteção* (*Protection Profiles - PP's*), *Alvos de Segurança* (*Security Targets - ST's*) e *pacotes* (combinação de requisitos de segurança específicos). Além disso, descreve a utilidade de cada uma das partes do CC para cada um dos seus grupos de usuários - clientes, desenvolvedores e avaliadores.

- **Parte 2 - Requisitos Funcionais de Segurança**

  A parte 2 contém um catálogo de requisitos funcionais de segurança (*functional requirements*) bem definidos e entendidos, que são usados como forma padronizada para expressar os requisitos de segurança de produtos e sistemas de tecnologias de informação, definindo o comportamento de segurança desejado.

- **Parte 3 - Requisitos de Garantia de Segurança**

  A parte 3 contém um catálogo de requisitos de garantia de segurança (*assurance requirements*), que pode ser usado como forma padronizada para expressar os requisitos de garantia, que são o fundamento para se conquistar a confiança de que as medidas de segurança solicitadas estão

[32] Um *catálogo* é definido por um grupo de componentes ou requisitos de segurança.
[33] Um *pacote* é uma combinação intermediária de componentes ou requisitos de segurança. O pacote permite a expressão de um conjunto de requisitos relacionados com um conjunto de objetivos de segurança. A idéia do pacote é que ele possa ser reutilizado e que defina requisitos que sejam úteis e efetivos para tratar os objetivos identificados.

efetiva e corretamente implementadas nos produtos e sistemas de tecnologia de informação. Esta terceira parte também define critérios de avaliação de *PP's* e *ST's*. Sete *Níveis de Garantia de Avaliação* ou *EALs* (*Evaluation Assurance Levels*) são apresentados nesta parte. Estes níveis são formados por conjuntos de componentes de garantia, pré-definidos, que habilitam o CC a classificar a confiança em produtos e sistemas na forma de uma escala crescente que vai do EAL1 até o EAL7.

6.1.2 Principais Conceitos

O CC define formas de expressar os requisitos de segurança de um sistema ou produto através de um conjunto de requisitos funcionais (ISO/IEC 15408-2, 1999) e de garantia (ISO/IEC 15408-3, 1999). Os requisitos funcionais (*functional requirements*) determinam o comportamento de segurança desejado para o sistema/produto, e seus requisitos tornam-se funções de segurança na implementação. Requisitos de garantia (*assurance requirements*) são o fundamento para se conquistar a confiança de que as funções de segurança estão efetiva e corretamente implementadas, satisfazendo os objetivos propostos.

Dentre alguns conceitos importantes desse critério estão: *TOE*, *TSP*, *TSF*, *PP*, *ST* e *Pacote*.

O **objeto de avaliação** (***TOE*** – *Target of Evaluation*) (Figura 6.1) é a entidade que é avaliada. O TOE pode ser um produto, uma parte de um produto, um conjunto de produtos, uma única tecnologia que nunca será um produto, ou uma combinação de todos estes fatores, em uma configuração específica ou um conjunto de configurações. Essa configuração ou conjunto de configurações específicas é chamada a *configuração avaliada.*

A **política de segurança de um TOE**, designada por ***TSP*** – *TOE Security Policy* – corresponde às regras que controlam como os recursos são gestionados, protegidos e distribuídos no interior de um TOE. As partes do TOE que implementam as políticas de segurança TSP e que devem ser confiáveis para que se tenha a aplicação correta das políticas são chamadas de *Funções de Segurança do TOE* (*TSF – TOE Security Functions*). As TSFs consistem de todo o *hardware*, *software* e *firmware* que está direta ou indiretamente afetando a concretização da segurança, e podem ser consideradas como a implementação do monitor de referência do TOE (capítulo 2) (ISO/IEC 15408-2, 1999). A Figura 6.1 apresenta um primeiro esboço dos requisitos funcionais de um TOE.

Alternativamente, conforme mostrado na Figura 6.2, um *TOE* pode ser um produto *distribuído* que consiste internamente de múltiplas partes separadas. Cada uma dessas partes fornece um serviço particular, e está conectada às outras partes do TOE através de um *canal de*

comunicação interno. Esse canal pode ser tão pequeno quanto um barramento de processador, ou pode englobar uma rede de comunicação interna ao TOE considerado.

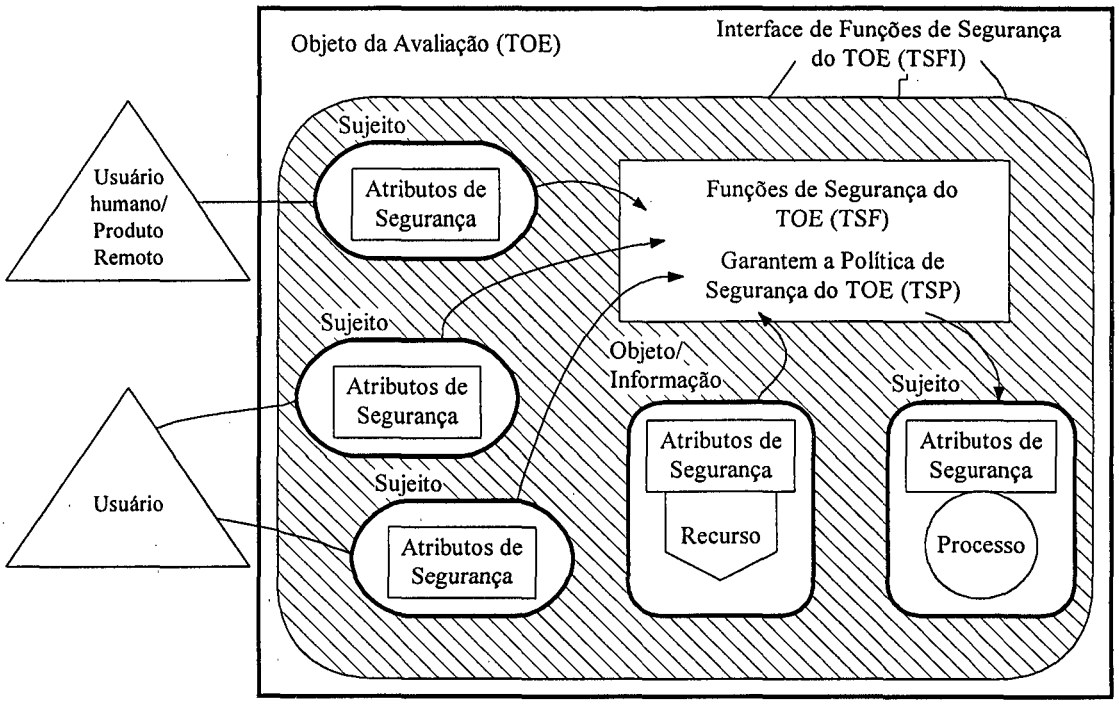


Figura 6.1 – Paradigma dos requisitos funcionais de segurança (TOE monolítico).

Quando um TOE consiste de múltiplas partes, onde cada parte pode conter sua própria parte das funções de segurança (SF), as funções de segurança do TOE interconectam-se e comunicam-se usando os canais internos. As interações envolvidas são chamadas *transferências internas do TOE*. As funções de segurança dos múltiplos componentes de um TOE distribuído compõem uma TSF global, que implementa a política de segurança TSP.

As interfaces do TOE podem interagir com outros produtos usando *canais de comunicação externos*. Essas interações externas com outros produtos podem ocorrer de duas formas:

- entre um TOE local e outro produto remoto confiável cujas políticas de segurança foram previamente avaliadas. Nessa situação as trocas de informação são chamadas *transferências inter-TSF*, pois ocorrem entre TSFs de produtos confiáveis distintos;
- entre um TOE local e outro produto remoto não confiável, visto que suas políticas de segurança não foram avaliadas e portanto são desconhecidas. Trocas de informação nessa situação são chamadas *transferências fora do controle da TSF*, já que não existe TSF (ou suas características de política são desconhecidas) no produto remoto.

A Figura 6.2 sintetiza essas definições.

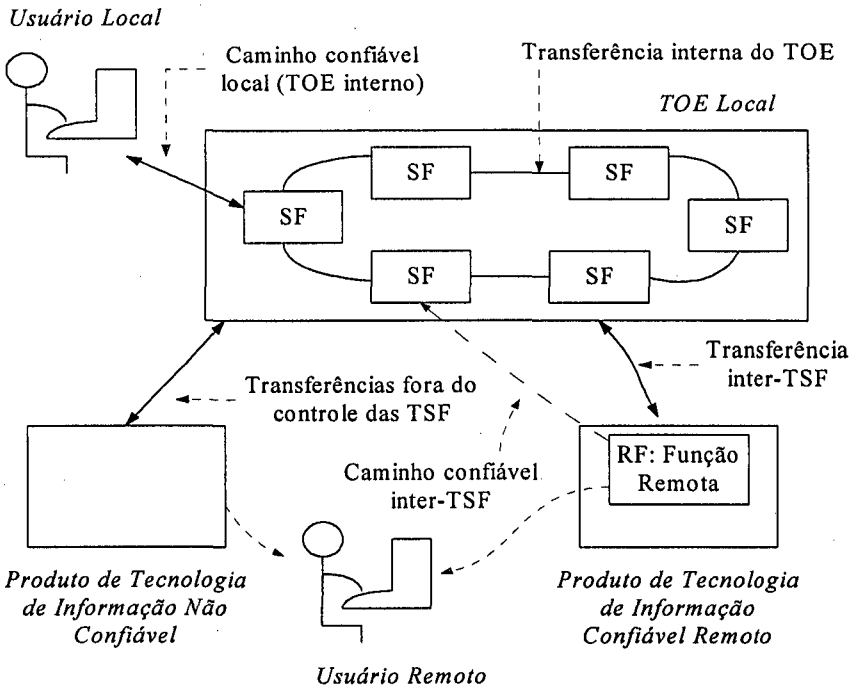


Figura 6.2 – Diagrama de funções de segurança em um TOE distribuído.

O **perfil de proteção** (***PP*** – *Protection Profile*) é uma definição de conjuntos de objetivos e requisitos, independente de implementação, para uma categoria de produtos ou sistemas de tecnologia da informação que tratam necessidades de segurança similares de consumidores (Figura 6.3). Exemplos de *profiles de proteção* são o *Controlled Access Protection Profile*, *Labeled Security Protection Profile*, *Traffic Filter Firewall Protection Profile for Low Risk Environments*, *Firewall-PP Draft*, *C2 Operating System Draft PP*, *RBAC-PP* (http://www.commoncriteria.org/protection_profiles/pp.html).

O **alvo de segurança** (***ST*** – *Security Target*) contém os requisitos de segurança de uma tecnologia de informação de um determinado TOE, e especifica as medidas funcionais e de garantia oferecidas por esse TOE para preencher os requisitos declarados (Figura 6.3).

O ST pode incorporar requisitos ou declarar conformidade a um ou mais PPs. O ST forma a *base para a avaliação de um TOE*, servindo como ponto de concordância entre desenvolvedores, avaliadores e, onde apropriado, consumidores sobre as propriedades de segurança do TOE e sobre o escopo da avaliação. Um ST descreve claramente a relação entre o TOE e quaisquer produtos associados (CEMEB, 1999). Dentre alguns STs estão: o ST para o Oracle V7 (MORRISSEY, 1998) e o ST para o Firewall Cisco (CSC, 1998) (http://niap.nist.gov/cc-scheme/ValidatedProducts.html).

A coleção dos requisitos reunida em um PP ou ST segue a seqüência de conteúdos da Figura 6.3. Maiores detalhes sobre cada um dos ítens são abordados na seção 6.3 e no Anexo 1, que descrevem a construção do ST para o projeto *JaCoWeb*.

Autoridades de Avaliação são responsáveis por certificar PPs e STs. Nos EUA, o esquema de avaliação é chamado NIAP (*National Information Assurance Partnership*) (http://niap.nist.gov/cc-scheme/) formado pelo NIST (*National Institute of Standards and Technology*) e NSA (*National Security Agency*) (NIST/NSA, 1999).

Protection Profile	***Security Target***
• Introdução	• Introdução
• Descrição do TOE	• Descrição do TOE
• Ambiente de Segurança	• Ambiente de Segurança
• Suposições	• Suposições
• Ameaças	• Ameaças
• Políticas de Segurança Organizacionais	• Políticas de Segurança Organizacionais
• Objetivos de Segurança	• Objetivos de Segurança
• Requisitos de Segurança	• Requisitos de Segurança
• Funcionais	• Funcionais
• De Garantia	• De Garantia
	• *Especificação Sumária do TOE*
	• *Declaração do PP (PP Claims)*
• Rationale (Interpretação)	• Rationale (Interpretação)

Figura 6.3 – Conteúdos de um PP e ST (ISO/IEC 15408-1, 1999).

Requisitos funcionais e de garantia dos PPs e STs, presentes em (ISO/IEC 15408-2, 1999) e (ISO/IEC 15408-3, 1999), são arranjados em uma hierarquia (Figura 6.4). Essa hierarquia é uma estratificação de construções que classificam os componentes dos requisitos de segurança em conjuntos relacionados: *classes*, *famílias* e *componentes*. Uma classe é um agrupamento de famílias que compartilham um foco comum – por exemplo, a classe *FDP User Data Protection* agrupa famílias que tratam da proteção dos dados do usuário. Uma família é um conjunto de componentes que compartilham objetivos de segurança mas podem diferir em ênfase ou rigor – por exemplo, a família *FDP_ACC Access Control Policy* identifica as diferentes políticas de controle de acesso. O componente é o menor conjunto de elementos que pode ser selecionado e que pode ser incluído em um PP/ST/*pacote*. Por exemplo, a família *FDP_ACC* é formada por dois componentes: *FDP_ACC.1*, que define que o controle de acesso pode ser aplicado somente a alguns sujeitos e objetos, e *FDP_ACC.2*, que define que o controle de acesso deve ser aplicado a todos os sujeitos e objetos do sistema. A Figura 6.4 mostra um exemplo de hierarquia.

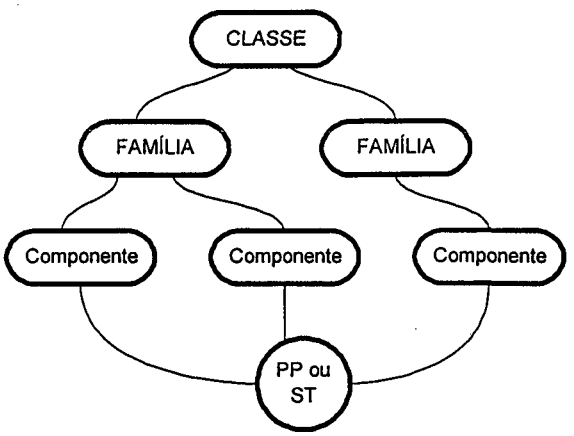


Figura 6.4 – Hierarquia de requisitos (TROY, 1999).

Um **pacote** é uma combinação intermediária de componentes. O pacote permite a expressão de um conjunto de requisitos funcionais e de garantia relacionados com um conjunto de objetivos de segurança. A idéia do pacote é que ele possa ser reutilizado, e que defina requisitos que sejam úteis e efetivos para tratar dos objetivos identificados. Um pacote pode ser utilizado na construção de pacotes maiores, PPs e STs.

O catálogo de requisitos funcionais (ISO/IEC 15408-2, 1999) é dividido em onze classes: auditoria (FAU), suporte criptográfico (FCS), comunicações (FCO), proteção de dados de usuário (FDP), identificação e autenticação (FIA), gerência da segurança (FMT), privacidade (FPR), proteção das funções de segurança do TOE (FPT), utilização de recursos (FRU), acesso ao TOE (FTA) e canais/caminhos confiáveis (FTP).

O catálogo de requisitos de garantia (ISO/IEC 15408-3, 1999) é formado por duas classes especiais contendo os requisitos de garantia para a avaliação de PPs e STs – classes *APE Protection Profile Evaluation* e *ASE Security Target Evaluation*, respectivamente. Neste catálogo existem ainda sete classes a partir das quais os requisitos de garantia podem ser escolhidos: classes de gerência de configuração (ACM), de entrega e operação (ADO), de desenvolvimento (ADV), de documentos guias (AGD), de suporte ao ciclo de vida (ALC), de testes (ATE) e de estimativa de vulnerabilidades (AVA). Ainda, outra classe que lida com a manutenção de garantia (AMA) é também parte do catálogo de requisitos de garantia.

O CC fornece garantia de segurança de um TOE usando o conceito de *investigação ativa* (ISO/IEC 15408-3, 1999), que é uma avaliação de um sistema ou produto de tecnologia da informação a fim de *determinar suas propriedades de segurança.* A garantia de segurança é o fundamento para se conquistar a confiança de que um produto ou sistema de tecnologia de informação atinge seus objetivos de segurança. Técnicas de avaliação que podem ser usadas são:

análise do projeto do TOE e de seus requisitos, testes funcionais, testes de penetração[34], análise de vulnerabilidades, dentre outras (ISO/IEC 15408-3, 1999).

A exemplo de padrões anteriores, o CC também possui um conjunto de níveis de garantia de segurança. Esses níveis de garantia (*assurance levels*) definem uma escala para medição das avaliações de PPs e STs. Os níveis EALs (*Evaluation Assurance Levels*) uniformizam o modo de balancear o nível de garantia obtido com o custo e a possibilidade de adquirir tal grau de garantia. Existem sete níveis de garantia: EAL1 (TOE testado funcionalmente), EAL2 (TOE testado estruturalmente), EAL3 (TOE testado metodicamente e verificado), EAL4 (TOE projetado metodicamente, testado e revisado), EAL5 (TOE projetado de maneira semi-formal e testado), EAL6 (TOE cujo projeto foi verificado de maneira semi-formal e testado), EAL7 (TOE cujo projeto foi verificado formalmente e testado). A Figura 6.5 representa a correspondência aproximada entre os níveis de garantia do CC e os níveis existentes nos critérios TCSEC e ITSEC. Mapeamentos exatos entre os níveis de garantia do CC e do TCSEC e ITSEC não existem já que o CC não define os níveis da mesma maneira que esses outros critérios.

Common Criteria	US TCSEC	ITSEC
-	D: Proteção Mínima	E0
EAL1		-
EAL2	C1: Proteção de Segurança Discricionária	E1
EAL3	C2: Proteção de Acesso Controlado	E2
EAL4	B1: Proteção de Segurança com Rótulos	E3
EAL5	B2: Proteção Estruturada	E4
EAL6	B3: Domínios de Segurança	E5
EAL7	A1: Projeto Verificado	E6

Figura 6.5 – Níveis de garantia do Critério Comum.

Os níveis de garantia de segurança (EALs) são pacotes de garantia pré-definidos contidos na parte 3 do CC. Um EAL é um conjunto *baseline*[35] de requisitos de garantia para avaliação. Cada EAL define um conjunto consistente de requisitos de garantia. Juntos, os EALs formam um conjunto ordenado de requisitos de garantia definindo uma escala de garantia pré-definida do CC.

6.1.3 Uso do CC

A abordagem do CC define como esses critérios aplicam os conceitos de segurança definidos. Quando do uso do padrão CC, a confiança na segurança de uma tecnologia de informação pode ser obtida através das ações realizadas durante o processo de *desenvolvimento*, *avaliação* e *operação* de um TOE.

[34] Testes de penetração (*penetration tests*) (KLEVINSKY, 1999) consistem no uso de ferramentas, usualmente manipuladas por *hackers* para atacar os sistemas, feito por usuários legítimos para a verificação de possíveis vulnerabilidades do sistema.
[35] *Baseline* pode ser definido como um "perfil de comportamento".

O processo de *desenvolvimento* está baseado no refinamento dos requisitos de segurança gerando uma especificação sumária de um TOE presente em um ST. A Figura 6.6 apresenta uma descrição deste processo de especificação sumária do TOE. Os critérios de garantia do CC identificam quatro níveis de refinamento no projeto de um TOE: a fase da especificação funcional, do projeto de alto-nível, do projeto de baixo-nível e da implementação. Dependendo do nível de garantia especificado, os desenvolvedores podem ser requisitados a mostrar como a metodologia de desenvolvimento trata dos requisitos de garantia do CC.

O processo de *avaliação* pode ser executado em paralelo com o desenvolvimento, ou pode ser a etapa seguinte. Maiores detalhes sobre a avaliação serão abordados no item 6.2.

Para a operação, o CC define que uma vez que um TOE esteja em operação, as vulnerabilidades podem surgir, ou as suposições sobre o ambiente operacional podem requerer revisões. Relatórios são então necessários para que o desenvolvedor modifique o TOE. Depois de feitas as modificações é necessário fazer uma reavaliação da segurança.

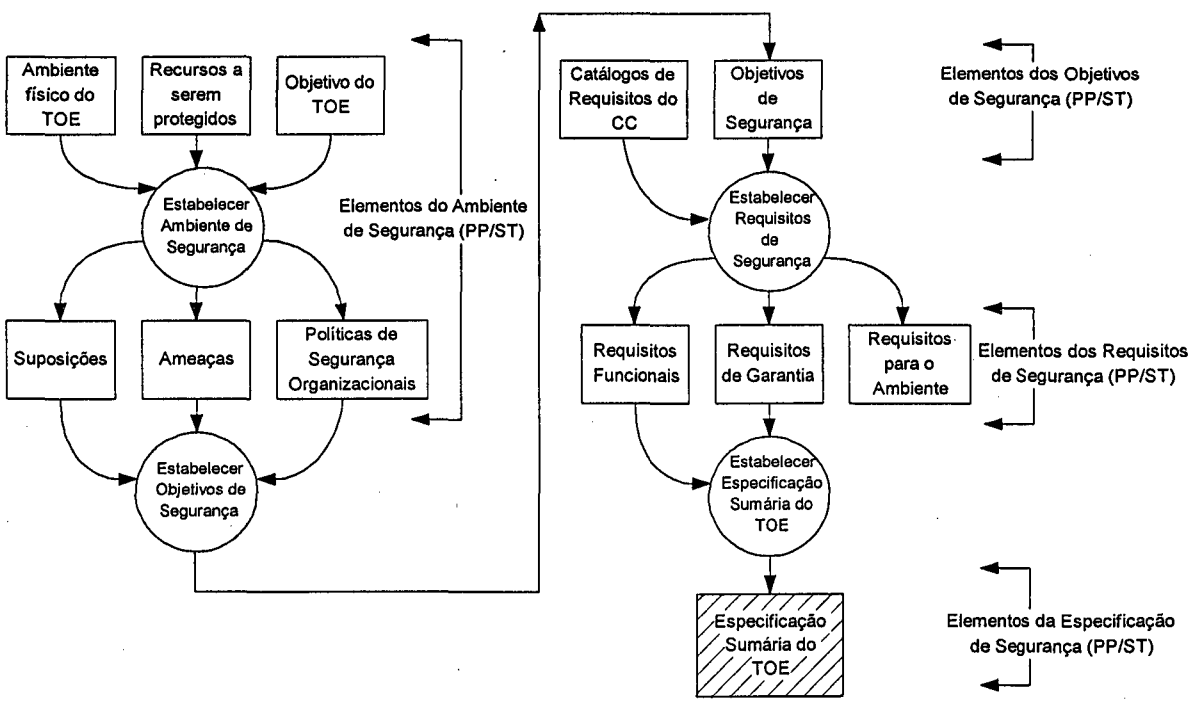


Figura 6.6 – Geração de Requisitos e Especificações (ISO/IEC 15408-1, 1999).

6.2 Avaliação de Produtos usando os Critérios Comuns

Usando o padrão Critérios Comuns (padrão ISO 15408), é possível realizar a avaliação de um *Protection Profile* (ISO/IEC 15408-3, 1999), ou a avaliação de um produto/sistema, que é composta de duas fases: a avaliação do *Security Target* (ISO/IEC 15408-3, 1999) e a avaliação do TOE (usando o *Security Target* como *baseline*) (Figura 6.7).

A avaliação do TOE inclui a *análise* do produto de tecnologia da informação e o *teste* do produto para verificar sua *conformidade* com um conjunto de requisitos de segurança. A avaliação pode melhorar o produto de duas formas: identificando erros ou vulnerabilidades no TOE que o desenvolvedor pode corrigir, reduzindo a probabilidade de erros futuros durante sua operação, e, também, impondo maior rigor no projeto e desenvolvimento do TOE.

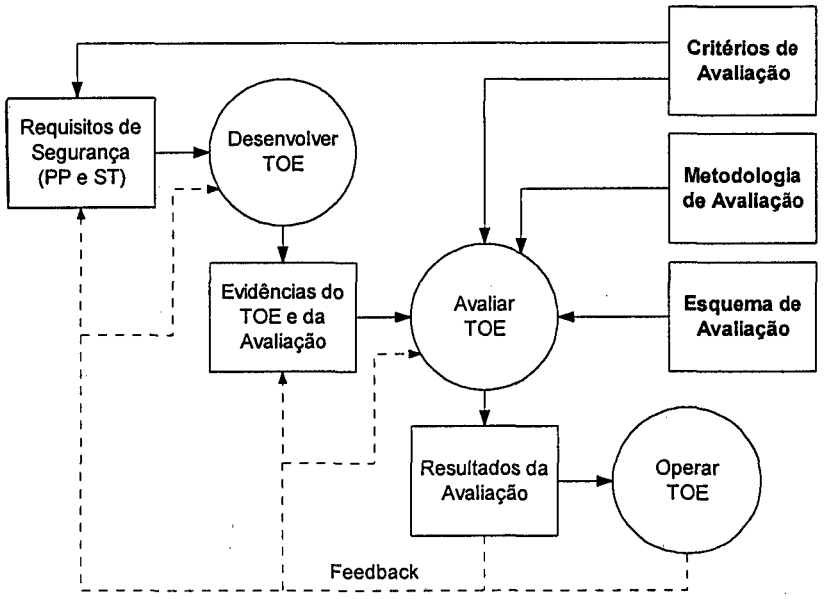


Figura 6.7 – Processo de Avaliação de um TOE (ISO/IEC 15408-1, 1999).

O padrão CC define que as principais entradas para a avaliação são: o conjunto de evidências, o próprio TOE, e os *critérios comuns* com a *metodologia* e o *esquema de avaliação* definidos. O conjunto de evidências é formado a partir de documentos de teste, documentos de projeto, códigos fontes, *hardware*, o ST avaliado ou qualquer outro recurso que possibilite ao avaliador obter informações sobre o TOE. O resultado esperado do processo de avaliação é uma confirmação de que o ST é satisfeito para o TOE, com um ou mais relatórios documentando a avaliação.

O *Esquema de Avaliação e Validação dos Critérios Comuns* (NIST/NSA, 1999) é o modo como é organizado todo o processo de avaliação e validação da segurança de um TOE segundo o CC (Figura 6.8). A avaliação de produtos de tecnologia de informação é feita com uma concordância entre laboratórios credenciados por uma entidade nacional responsável. Essa entidade nacional é responsável pela validação da avaliação realizada, emitindo certificados de garantia. Esses certificados de garantia são reconhecidos pelas nações participantes do *MRA* (*Mutual*

Recognition Arrangement)[36]. *Validação*, nesse contexto, representa a revisão da avaliação com o objetivo de identificar se a mesma foi conduzida de acordo com os critérios do esquema, e de verificar que as conclusões do laboratório que realizou os testes são consistentes com os fatos apresentados na avaliação. Nos USA, o esquema de avaliação e validação é designado *NIAP Common Criteria Evaluation and Validation Scheme for IT Security* (NIST/NSA, 1999) e é uma parceria entre setores públicos (NIST+NSA) e privados.

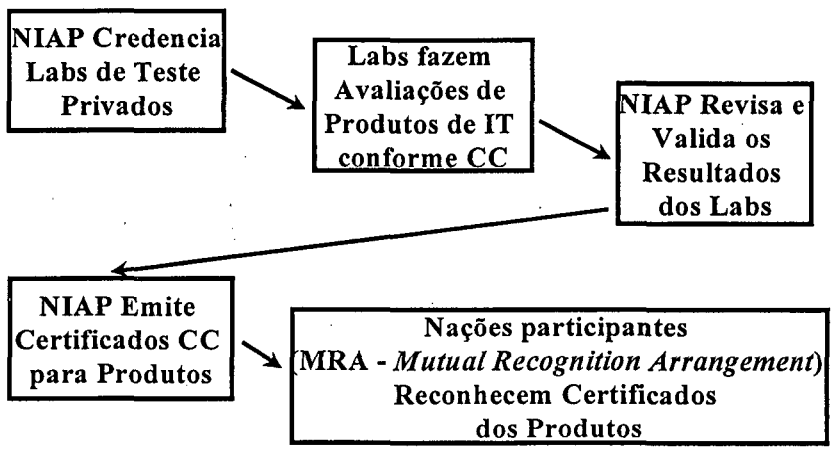


Figura 6.8 – *Esquema* de Avaliação dos USA (NIST/NSA, 1999, TROY, 1999).

A *Metodologia de Avaliação Comum* (*CEM – Common Evaluation Methodology*) (CEMEB, 1997, CEMEB, 1999) é um documento necessário para realizar a avaliação usando o CC. Esse documento descreve as ações mínimas a serem executadas por um avaliador a fim de conduzir uma avaliação, usando os critérios e evidências de avaliação definidas no CC.

O escopo da versão atual dessa metodologia é limitada a avaliações de *Protection Profiles* e *TOEs* do nível EAL1 ao nível EAL4, conforme definido no CC.

O público alvo dessa metodologia é principalmente os avaliadores que aplicam o CC (e as entidades de certificação que confirmam as ações dos avaliadores), os requisitantes das avaliações, os desenvolvedores, os autores de PP/STs e outras partes interessadas na publicação e uso dos resultados de avaliação.

A CEM é usada em *esquemas de avaliação* para garantir a *aplicação consistente* dos requisitos do CC em múltiplas avaliações. Portanto, a CEM é um componente importante do reconhecimento mútuo internacional *MRA*.

[36] Treze nações são membros do MRA (http://www.commoncriteria.org/registry/ccra-final.html): Canadá, França, Alemanha, Inglaterra, Nova Zelândia, Austrália, Finlândia, Grécia, Itália, Noruega, Holanda, Espanha e USA (NIAP/NIST, 1998). A idéia do MRA é reconhecer as avaliações que recebem os respectivos certificados em qualquer um dos países participantes. Dentre os objetivos do MRA, encontram-se: garantir avaliações de produtos e PPs feitas com padrões consistentes que contribuam significativamente para a confiança na segurança desses produtos e PPs; aumentar a disponibilidade de produtos e PPs avaliados para uso internacional; eliminar avaliações repetidas e melhorar continuamente a eficiência e custo efetivo das avaliações e o processo de certificação/validação de produtos e PPs. A idéia é que novos países sejam adicionados ao MRA no futuro (TROY, 1999).

Todas as avaliações, tanto de PPs quanto de TOEs ou STs, possuem em comum *tarefas de entrada* e *tarefas de saída* (CEMEB, 1999).

As *tarefas de entrada* consistem na obtenção, por parte do avaliador, de evidências necessárias para avaliação. A exatidão técnica da avaliação deve ser assegurada pela forma de condução bem definida de modo que os resultados possam ser repetidos ou reproduzidos.

As *tarefas de saída* consistem na elaboração de relatórios de observação (*OR – Observation Reports*), quando necessários no contexto da avaliação, e de relatórios técnicos de avaliação (*ETR – Evaluation Technical Reports*). Um OR fornece ao avaliador uma forma de pedir uma explicação para identificar um problema com relação a avaliação. Um ETR deve ser fornecido por um avaliador para justificar tecnicamente os vereditos atribuídos ao TOE. Nosso objetivo aqui será a elaboração de um ETR final para o projeto *JaCoWeb* (seção 6.4). Um ETR, a exemplo de PPs e STs, define um conteúdo preciso diante da metodologia de avaliação (CEMEB, 1999) (Figura 6.9). Cada um dos ítens presentes no ETR será descrito na seção 6.4.

Evaluation Technical Report
- **Introdução**
- **Descrição da Arquitetura do TOE**
- **Avaliação**
- **Resultados da Avaliação**
- **Conclusões e Recomendações**
- **Lista de Evidências da Avaliação**
- **Lista de acrônimos/Glossário de termos**
- **Relatórios de Observação**

Figura 6.9 – Conteúdo de um ETR na avaliação de um TOE (CEMEB, 1999).

Existem dois estágios para a avaliação de um TOE (CEMEB, 1999): a avaliação do ST e a avaliação do próprio TOE. A avaliação do ST é feita antes ou em paralelo à avaliação do TOE. O objetivo da avaliação do ST é determinar:

- Se o ST é *completo*, ou seja, cada ameaça é impedida e cada política de segurança definida é aplicada pelas funções de segurança;
- Se o ST é *suficiente*, ou seja, as funções de segurança são apropriadas para as ameaças e políticas de segurança definidas, e se as medidas de garantia fornecem uma certeza de que as funções de segurança estão implementadas corretamente;
- Se o ST é *correto*, ou seja, o ST deve ser internamente coerente e contínuo;
- Se o ST é *instanciado de maneira correta*, ou seja, se o ST declara conformidade com um ou mais PPs, então o ST deve instanciar de forma exata e completa cada um dos PPs referenciados.

A Figura 6.10 mostra as etapas que compõem uma avaliação completa do ST.

- **Definição de Tarefas de entrada;**
- **Atividade de avaliação do ST, que compreende as seguintes sub-atividades:**
  - **Avaliação da Descrição do TOE (ASE_DES);**
  - **Avaliação do Ambiente de Segurança (ASE_ENV);**
  - **Avaliação da Introdução ao ST (ASE_INT);**
  - **Avaliação dos Objetivos de Segurança (ASE_OBJ);**
  - **Avaliação do PP *Claims* (ASE_PPC);**
  - **Avaliação dos Requisitos de Segurança da IT (ASE_REQ);**
  - **Avaliação do Requisitos de Segurança explicitamente definidos (ASE_SRE);**
  - **Avaliação da Especificação Sumária do TOE (ASE_TSS);**
- **Tarefas de saída.**

Figura 6.10 – Etapas da avaliação do ST.

As sub-atividades de avaliação do ST são derivadas da classe ASE dos requisitos de garantia contidos na parte 3 do CC (ISO/IEC 15408-3, 1999). A avaliação do *JaCoWeb-ST* construído para o esquema de autorização considerado é descrita no Anexo 2, e é feita em paralelo com a avaliação do TOE.

A seguir iniciamos o processo de avaliação propriamente dito, construindo um ST (seção 6.3) onde então possamos avaliar o *JaCoWeb-ST*, gerando no final o relatório técnico ETR (seção 6.4) no sentido de documentar a avaliação do esquema de autorização *JaCoWeb*.

6.3 *JaCoWeb-ST (Security Target)*

O *JaCoWeb-ST* contém os requisitos de segurança do TOE Esquema de Autorização *JaCoWeb DiscMand* (Discricionário e Obrigatório), Versão 1.0, especificando as medidas funcionais e de garantia oferecidas pelo *JaCoWeb* para preencher os requisitos declarados. O TOE *JaCoWeb* está em conformidade com o *Protection Profile* designado *Labeled Security Protection Profile* (*LSPP*), Version 1.b, 8 October 1999 (NSA, 1999), registrado pela Agência de Segurança Nacional dos Estados Unidos (NSA – *National Security Agency*).

Nosso objetivo nessa seção é apresentar algumas considerações sobre aspectos mais importantes do *JaCoWeb-ST*. O *JaCoWeb-ST*, feito de acordo com o formato estabelecido pelo padrão dos Critérios Comuns, está disponível no Anexo 1.

Um documento ST fornece a *base para a avaliação* de um produto ou sistema de tecnologia de informação (*IT – Information Technology*). Um ST define principalmente:

- um conjunto de suposições sobre os aspectos de segurança do ambiente, uma lista de ameaças que o produto pretende conter e regras ou políticas que o produto deve suportar;
- um conjunto de objetivos de segurança e um conjunto de requisitos de segurança para tratar o problema;

– as funções de segurança da IT fornecidas pelo TOE que satisfazem o conjunto de requisitos.

A descrição de algumas destas definições em relação ao TOE *JaCoWeb* é apresentada a seguir.

6.3.1 Características de Segurança do Esquema de Autorização JaCoWeb DiscMand Versão 1.0

O TOE *JaCoWeb* fornece as seguintes características de segurança:

– **Auditoria**: A geração de dados de auditoria são feitos via arquivos de registros (*logs*) e implementados conforme política de auditoria introduzida no modelo CORBAsec. Esta política é definida pelo administrador do sistema.

– **Suporte Criptográfico**: Os controles criptográficos são implementados pelo uso do *iSaSiLk* (GRAZ, 1999a), versão do SSL em Java, durante o estabelecimento da conexão segura usando o CORBA. O pacote IAIK-JCE (GRAZ, 1999b) implementa os algoritmos criptográficos usados pelo *iSaSiLk*. O algoritmo de chave pública, usado para autenticação mútua e distribuição de chaves, é o RSA de 1024 bits. Para a cifragem dos dados é usado o algoritmo simétrico 3DES_EDE (cifragem em blocos) e a função *hash* de segurança SHA para a computação do MAC (capítulo 4). A assinatura dos *applets*[37] clientes é feita usando recursos da linguagem Java JDK 1.2.1.

– **Proteção de Dados do Usuário**: As políticas de proteção de dados do usuário são implementadas a partir dos objetos de política de segurança do sistema, centralizados no serviço de política *PoliCap*. Além desses objetos, os objetos locais de política e os objetos de decisão de acesso e de sessão (*PolicyCurrent*, *Current* e *SecurityManager*) implementam as funções de controle de acesso do sistema, localmente em cada máquina que participa das comunicações. A autenticação mútua entre as partes comunicantes é estabelecida através do estabelecimento da associação segura via CORBA, usando o SSL, entre objetos de aplicação cliente e servidor. A cada nova associação segura estabelecida entre dois objetos quaisquer, novos objetos de serviço de segurança são instanciados por sessão, impedindo que qualquer informação residual (prévia/permanente) seja transferida em novas sessões.

[37] Um relatório do processo de assinatura do *applet* cliente está disponível no endereço http://www.lcmi.ufsc.br/jacoweb/documentos.

- **Identificação e Autenticação**: Há três fases de autenticação presentes no TOE: a autenticação *do código móvel ou applet Java*, a autenticação do *principal* e a autenticação *mútua entre cliente e servidor* (autenticação da sessão do modelo de segurança CORBAsec). A autenticação do *applet* Java, que é carregado pelo *browser* e representa o cliente no ambiente considerado, usa os recursos de assinatura de *applets* provido pelo ambiente Java JDK 1.2. A *autenticação do principal* pelo modelo CORBAsec é realizada pelo pacote *JaCoWebSecurity*, usando certificados X.509v3 dos usuários. Estes certificados também são usados para a autenticação mútua entre cliente e servidor (*autenticação da sessão* do modelo CORBAsec), que é feita pelo próprio SSL, durante o estabelecimento da associação segura do CORBAsec (capítulo 4).
- **Gerência da Segurança**: A gerência dos objetos de política e da relação entre usuários e atributos de privilégio é realizada por administradores autorizados do sistema. Objetos da aplicação também atuam para definir os objetos de política na parte que lhes cabe.
- **Proteção das Funções de Segurança**: As funções de segurança não podem ser evitadas ou modificadas, pois as políticas de segurança são aplicadas de forma transparente a uma aplicação pelo próprio ORB, através dos objetos de serviço do CORBAsec, presentes no pacote *JaCoWebSecurity*.

O TOE é usado para prover programação distribuída segura em ambientes de pequena e larga escala. Possui interface gráfica representada pelo *applet* de aplicação instalada na máquina do cliente. Esse *applet* é carregado a partir de um servidor Web. O serviço de política e servidores de aplicação são instanciados no momento em que um cliente inicia requisições de serviços. O pacote *JaCoWebSecurity* fornece acesso aos objetos de serviço do CORBAsec implementados.

6.3.2 Ameaças tratadas pelo Esquema de Autorização JaCoWeb DiscMand Versão 1.0

O TOE foi concebido para tratar as seguintes ameaças:

Ameaça 1) Um usuário autorizado do TOE tenta acessar informações ou recursos sem ter permissão do proprietário ou responsável pela informação ou recurso.

Ameaça 2) Um invasor tenta observar ou capturar dados sendo transmitidos pela rede.

Ameaça 3) A integridade da informação pode ser comprometida devido a erros do usuário, de transmissão ou falhas no *hardware*.

Ameaça 4) Um invasor (interno ou externo) tenta obter acesso autorizado a informações ou recursos, se fazendo passar por um usuário autorizado do TOE.

Ameaça 5) O TOE pode ser inadvertidamente configurado, usado e administrado de forma insegura por pessoas autorizadas ou não autorizadas.

6.3.3 Políticas de Segurança Organizacionais

Uma política de segurança organizacional é um conjunto de regras ou procedimentos impostos por uma organização sobre operações para proteger seus dados sensíveis. As políticas organizacionais descritas nesse item estão em conformidade com o LSPP e são descritas na Tabela 6.1.

Nome	Descrição
P.AUTHORIZED_ USERS	Apenas usuários autorizados a usar as informações do sistema têm acesso ao sistema.
P.NEED_TO_KNOW	O sistema deve limitar o acesso, a modificação e a destruição da informação, em recursos protegidos, aos usuários autorizados.
P.ACCOUNTABILITY	Os usuários do sistema devem ser responsabilizados por suas ações dentro do sistema.
P.CLASSIFICATION	O sistema deve regrar as formas de acesso, tomando como base os rótulos de sensibilidade das informações contidas em objetos e a habilitação dos usuários, representada pelos sujeitos que requisitam os acessos correspondentes. As regras de acesso aplicadas impedem o sujeito de ter acesso a informações com sensibilidade maior do que a sua habilitação, e impedem também o sujeito de causar uma modificação de informações em níveis de sensibilidade inferiores. A determinação da classificação da informação e da habilitação dos usuários está fora do escopo do sistema de tecnologia de informação. Essa classificação é essencialmente baseada em regras da organização e nas funções exercidas pelos usuários dentro da organização.
P.DAC	O direito de acesso a objetos específicos é determinado tomando como base a identidade do sujeito que deseja o acesso e os direitos de acesso implícitos e explícitos do objeto, fornecidos ao sujeito pelo proprietário do objeto.

Tabela 6.1 – Política de Segurança Organizacional.

6.3.4 Objetivos de Segurança

Esta seção define os objetivos de segurança das funções de segurança do TOE *JaCoWeb* e do seu ambiente de suporte. O propósito dos objetivos de segurança é detalhar uma resposta planejada aos problemas ou ameaças à segurança do sistema. Ameaças podem ser direcionadas contra o TOE ou o ambiente de segurança ou ainda contra ambos. Os objetivos de segurança do TOE são descritos na Tabela 6.2.

Nome	Descrição
O.AUTHORIZATION	As funções de segurança devem garantir o acesso ao TOE e aos seus recursos apenas a usuários autorizados.
O.DISCRETIONARY_ ACCESS	As funções de segurança devem controlar o acesso aos recursos baseadas na identidade dos usuários. As funções de segurança devem permitir aos usuários autorizados a possibilidade de especificar quais recursos podem ser acessados por quais usuários.
O.MANDATORY_ ACCESS	As funções de segurança devem controlar o acesso aos recursos baseadas na sensibilidade e categorias da informação que está sendo acessada, e na habilitação do sujeito que requisita o acesso à informação.
O.AUDITING	As funções de segurança devem registrar as ações de segurança relevantes dos usuários do TOE. As funções de segurança devem apresentar essas informações para administradores autorizados.
O.RESIDUAL_ INFORMATION	As funções de segurança devem garantir que informações contidas em um recurso protegido não sejam liberadas quando o recurso é reutilizado (*recycled*).
O.MANAGE	As funções de segurança devem fornecer todas as funções e facilidades necessárias para suportar administradores autorizados que sejam responsáveis pelo gerenciamento da segurança do TOE.
O.ENFORCEMENT	As funções de segurança devem ser projetadas e implementadas de forma a garantir que as políticas de segurança organizacionais sejam aplicadas no ambiente destino.

Tabela 6.2 – Objetivos de Segurança do TOE *JaCoWeb*.

6.3.5 Requisitos de Segurança da Tecnologia de Informação

Os requisitos de segurança do TOE concretizam os objetivos de segurança. O CC divide os requisitos de segurança do TOE em duas categorias:

- Requisitos funcionais de segurança (*SFRs – Security Functional Requirements*), isto é, requisitos para as funções de segurança, tais como controle de acesso às informações, auditoria, identificação e autenticação.
- Requisitos de garantia de segurança (*SARs – Security Assurance Requirements*), que fornecem a base para a confiança de que o TOE atinge os seus objetivos de segurança (por exemplo, documentos guias, testes, declaração de vulnerabilidades).

6.3.5.1 Requisitos Funcionais de Segurança do TOE

O catálogo de requisitos funcionais (ISO/IEC 15408-2, 1999) é dividido em onze classes: auditoria (FAU), suporte criptográfico (FCS), comunicações (FCO), proteção de dados de usuário (FDP), identificação e autenticação (FIA), gerência da segurança (FMT), privacidade (FPR),

proteção das funções de segurança do TOE (FPT), utilização de recursos (FRU), acesso ao TOE (FTA) e canais/caminhos confiáveis (FTP).

Por exemplo, a Tabela 6.3 apresenta a classe FDP que trata da Proteção dos Dados dos Usuários. Essa classe contêm famílias especificando os requisitos das funções de segurança do TOE e das políticas de funções de segurança relacionadas ao processamento de dados de usuário. Essa classe é dividida em quatro grupos de famílias que tratam dos dados de usuário dentro do TOE, durante a importação, exportação e armazenamento dos mesmos, bem como dos atributos de segurança diretamente relacionados aos dados do usuário.

Nome da Classe	**Identificador do Componente Funcional**	**Nome do Componente Funcional**
Classe FDP – Proteção dos Dados dos Usuários	FDP_ACC.1	Política de Controle de Acesso Discricionária
	FDP_ACF.1	Funções de Controle de Acesso Discricionárias
	FDP_ETC.1	Exportação de Dados de Usuário Não Rotulados
	FDP_ETC.2	Exportação de Dados de Usuário Rotulados
	FDP_IFC.1	Política de Controle de Acesso Obrigatória
	FDP_IFF.2	Funções de Controle de Acesso Obrigatórios
	FDP_ITC.1	Importação de Dados de Usuário Não-Rotulados
	FDP_ITC.2	Importação de Dados de Usuário Rotulados
	FDP_RIP.2	Proteção às Informações Residuais dos Objetos

Tabela 6.3 – Classe FDP.

Seguindo então este catálogo de requisitos funcionais, o TOE *JaCoWeb* possui as seguintes classes:

Gerência da Segurança (FMT)

O serviço de política *PoliCap* mantém os atributos de segurança do ambiente para todos os administradores, usuários e objetos do sistema. As interfaces *DomainAccessPolicy* e *RequiredRights* oferecem operações que são usadas por aplicações administrativas para a definição da política de segurança discricionária e/ou obrigatória do domínio (objeto *DomainAccessPolicy*) e do objeto de direitos requeridos (objeto *RequiredRights*).

O objeto de política de auditoria *Audit Policy* determina os dados a serem examinados durante uma sessão. Esse objeto é definido pelo administrador autorizado do sistema.

As seguintes funções administrativas podem ser executadas por usuários autenticados e autorizados:

- criar, remover, modificar e verificar os atributos dos usuários e dos objetos;
- modificar e definir o número de tentativas de autenticação permitidas por administradores a usuários normais do TOE;

- restabelecer a capacidade de autenticação para usuários que ultrapassaram as tentativas de autenticação sem sucesso;
- arquivar, criar, remover, revisar e esvaziar a trilha (*trail*) de auditoria.

Os **requisitos funcionais satisfeitos no *JaCoWeb*** em relação à gerência da segurança (ver Tabela 1.15 do Anexo 1) são: FMT_MSA.1, FMT_MSA.3, FMT_MTD.1 (1), FMT_MTD.1 (2), FMT_MTD.1 (3), FMT_MTD.1 (4), FMT_REV.1 (1), FMT_REV.1 (2) e FMT_SMR.1.

Identificação e Autenticação (FIA)

A identificação e autenticação tratam da definição de atributos do usuário, da 'força' dos dados de autenticação, da autenticação, da proteção dos dados usados no momento da autenticação, da identificação e da ligação entre usuário e sujeito.

As três fases de autenticação presentes no TOE *JaCoWeb* - a autenticação *do código móvel ou applet Java*, a autenticação do *principal* e a autenticação *mútua entre cliente e servidor* - já foram discutidas no capítulo 4 e brevemente na seção 6.3.1. A autenticação do *applet* Java usa os recursos de assinatura de *applets* provido pelo ambiente Java JDK 1.2. A *autenticação do principal* pelo modelo CORBAsec é realizada pelo pacote *JaCoWebSecurity*, usando certificados X.509v3 dos usuários. Estes certificados também são usados para a autenticação mútua entre cliente e servidor (*autenticação da sessão* do modelo CORBAsec), que é feita durante o estabelecimento da associação segura do CORBAsec.

A proteção dos dados durante o processo de autenticação do principal, a fim de gerar as credenciais, é fornecida pela classe *Security*, que impede que a aplicação acesse dados durante a fase de autenticação.

A 'força' dos dados de autenticação é medida, verificando-se que existe baixa probabilidade de que os dados usados na autenticação possam ser forjados ou adivinhados. Nesse TOE, os dados de autenticação são fornecidos através de certificados X.509v3 fornecidos por uma terceira entidade (ITU-TX509, 1993).

Os **requisitos funcionais satisfeitos no *JaCoWeb*** em relação à identificação e autenticação (ver Tabela 1.14 do Anexo 1) são: FIA_ATD.1, FIA_SOS.1, FIA_UAU.1, FIA_UAU.7, FIA_UID.1 e FIA_USB.1.

Proteção de Dados de Usuário (FDP)

Os requisitos para a proteção dos dados do usuário são compostos pela política de controle de acesso discricionária, pela política de controle de acesso obrigatória, pela importação de dados de usuário rotulados e não-rotulados, pela exportação de dados de usuário rotulados e não-rotulados, e pela proteção das informações residuais dos objetos.

As políticas de controle de acesso discricionária e obrigatória são implementadas pelo serviço de política *PoliCap*, descrito no capítulo 4 e no capítulo 5, que usa os objetos de serviço do CORBAsec. As funções de controle de acesso discricionárias e obrigatórias, que são responsáveis pela autorização da invocação, são implementadas pelos objetos *AccessDecision* locais em cada máquina participante da aplicação distribuída.

A importação e a exportação de dados relativos ao usuário, rotulados e não-rotulados, é controlada pelos objetos de política do CORBAsec no sistema.

A cada nova associação segura estabelecida entre dois objetos quaisquer, novos objetos de sessão são instanciados, impedindo que qualquer informação prévia (residente) seja transferida para futuras sessões.

Os **requisitos funcionais satisfeitos no *JaCoWeb*** em relação à proteção de dados de usuário (ver Tabela 1.13 do Anexo 1) são: FDP_ACC.1, FDP_ACF.1, FDP_ETC.1, FDP_ETC.2, FDP_IFC.1, FDP_IFF.2, FDP_ITC.1, FDP_ITC.2 e FDP_RIP.2.

Proteção das Funções de Segurança (FPT)

A proteção das funções de segurança TSFs (*TOE Security Functions*) é necessária para fornecer requisitos que garantam a não modificação e evitem o *bypass* (a não execução), das funções que implementam as políticas de segurança do TOE.

Do ponto de vista desta classe, existem três partes significativas que compõem as TSFs: a máquina abstrata das TSFs, a implementação e os dados das TSFs. A máquina abstrata das TSFs é definida pela máquina física ou virtual sobre a qual é executada a implementação de uma TSF específica sob avaliação. A implementação das TSFs concretiza os mecanismos que garantem as políticas de segurança do TOE, e é executada sobre a máquina abstrata. Os dados das TSFs são bancos de dados administrativos que guiam a aplicação da política.

Os requisitos para a proteção das funções de segurança TSFs, de acordo com esta classe, são compostos pelo "teste da máquina abstrata", pela mediação de referência, pela separação de domínios e por *time-stamps* confiáveis. Todos esses requisitos servem para garantir a integridade e gerência dos mecanismos que fornecem as TSFs e garantir a integridade dos dados das TSFs.

O teste da máquina abstrata diz respeito a garantir que, durante a inicialização do sistema, ou na sua operação normal ou ainda durante os pedidos de usuários autorizados, a máquina abstrata sobre a qual o TOE é executado opere de forma correta, sem quaisquer falhas de projeto. Os cuidados no desenvolvimento e operação do TOE *JaCoWeb*, com o uso de ferramentas de especificação e de metodologias de construção de *software*, nos permite assumir esta premissa de que nada de anormal ocorra para cobrir esse requisito funcional (seção 6.3.1).

O requisito funcional da mediação de referência trata o aspecto tradicional do monitor de referência, garantindo que todas as ações a serem verificadas de acordo com a política de segurança, sejam validadas pelas TSFs que implementam as políticas de segurança do ambiente.

No *JaCoWeb*, as TSFs, que são implementadas pelos objetos *AccessDecision* locais do CORBAsec em cada *site* do ambiente, garantem que as políticas de segurança não podem ser evitadas durante uma requisição da aplicação. A política de segurança aplicada a nível de ORB, como é o caso desse TOE, é executada automaticamente pelo próprio ORB de forma transparente para a aplicação, através dos objetos de serviço do CORBAsec, presentes no pacote *JaCoWebSecurity*.

A separação de domínios diz respeito ao isolamento das TSFs, em relação às aplicações ou outros sujeitos não confiáveis, para prevenir a modificação de códigos ou estruturas de dados e a interferência durante a execução das TSFs. No *JaCoWeb*, este requisito funcional é implementado pelo uso dos interceptadores do modelo CORBAsec, impossibilitando a invocação das TSFs à sujeitos não confiáveis.

O requisito funcional dos *time-stamps*, que são usados pelas próprias TSFs, é necessário caso existam eventos de auditoria que são registrados pelo sistema. Neste caso, devem existir, no mínimo, notificações do relógio, de forma a atribuir *time-stamps* a estes eventos. No *JaCoWeb*, os *time-stamps* são implementados pelo uso do relógio do sistema, a fim de registrar os eventos de auditoria.

Os **requisitos funcionais satisfeitos no *JaCoWeb*** em relação à proteção das funções de segurança (ver Tabela 1.16 do Anexo 1) são: FPT_AMT.1, FPT_RVM.1, FPT_SEP.1 e FPT_STM.1.

Funções Criptográficas (FCS)

Os requisitos que são usados para definir as funções criptográficas são o gerenciamento de chaves criptográficas e a operação criptográfica propriamente dita.

Estas operações necessárias durante o estabelecimento e a duração de uma conexão segura no CORBAsec são providas pelo *iSaSiLk* (GRAZ, 1999a), versão do SSL em Java. Os algoritmos

disponíveis via *iSaSiLk* são implementados no pacote IAIK-JCE (GRAZ, 1999b). O algoritmo de chave pública, usado para autenticação mútua e distribuição de chaves, é o RSA de 1024 bits. Na cifragem dos dados utiliza-se o algoritmo simétrico de cifragem em bloco 3DES_EDE e a função *hash* de segurança SHA para a computação do MAC (capítulo 4). A assinatura dos *applets* clientes é feita usando recursos da linguagem Java JDK 1.2.1.

Os **requisitos funcionais satisfeitos no *JaCoWeb*** em relação à proteção das funções de segurança (ver Tabela 1.17 do Anexo 1) são: FCS_CKM.3, FCS_CKM.4 e FCS_COP.1.

Auditoria (FAU)

Os requisitos de auditoria contemplam a geração dos dados de auditoria, a associação da identidade do usuário, a revisão da auditoria, a auditoria seletiva, as garantias de disponibilidade de dados de auditoria, a ação no caso de perdas dos dados de auditoria e a prevenção da perda dos dados de auditoria.

A geração de dados de auditoria são feitos via arquivos de *logs*, e implementados pela política de auditoria (objeto *AuditPolicy*) definida no modelo CORBAsec. Essa política é definida pelo administrador do sistema, e é armazenada no serviço de políticas *PoliCap,* sendo carregada em tempo de ligação entre objetos cliente e servidor em ambos os lados.

Aplicações administrativas fornecem acesso aos dados de auditoria apenas para o administrador do sistema, que pode revisar e selecionar dados de interesse. Depois de um certo tempo, o administrador é responsável por eliminar dados desnecessários dos arquivos de *log* gerados.

A auditoria no *JaCoWeb* usa os objetos de serviço do CORBAsec *AuditDecision*, *AuditPolicy* e *AuditChannel* (BLAKLEY, 1999, OMG, 2000a) e foi detalhada no capítulo 4.

Os **requisitos funcionais satisfeitos no *JaCoWeb*** em relação à auditoria (ver Tabela 1.12 do Anexo 1) são: FAU_GEN.1, FAU_GEN.2, FAU_SAR.1, FAU_SAR.2, FAU_SAR.3, FAU_SEL.1, FAU_STG.1, FAU_STG.3 e FAU_STG.4.

6.3.5.2 Requisitos de Garantia (assurance) de Segurança do TOE JaCoWeb

A Tabela 6.4 identifica os componentes de garantia de segurança (*Security Assurance Requirements*) presentes no *JaCoWeb*, que estão de acordo com a parte 3 do CC (ISO/IEC 15408-3, 1999), perfazendo o nível EAL 3 e estando em conformidade com LSPP (*Labeled Security Protection Profile*) (NSA, 1999).

Nome da Classe de Garantia	Identificador do Componente de Garantia	Nome do Componente de Garantia
Classe ACM – Gerência de Configuração	ACM_CAP.3	Controles de Autorização
	ACM_SCP.1	Cobertura da Gerência de Configuração do TOE
Classe ADO – Liberação (*delivery*) e Operação	ADO_DEL.1	Procedimentos de liberação
	ADO_IGS.1	Instalação, geração e procedimentos iniciais
Classe ADV – Desenvolvimento	ADV_FSP.1	Especificação funcional informal
	ADV_HLD.2	Aplicação da segurança no projeto de alto-nível
	ADV_RCR.1	Demonstração da correspondência informal
	ADV_SMP.1	Modelagem da política de segurança
Classe AGD – Documentos Guias	AGD_ADM.1	Guia do administrador
	AGD_USR.1	Guia do usuário
Classe ALC – Suporte ao Ciclo de Vida	ALC_DVS.1	Identificação das medidas de segurança
Classe ATE – Testes	ATE_COV.2	Análise de cobertura
	ATE_DPT.1	Testes: projeto de alto nível
	ATE_FUN.1	Testes funcionais
	ATE_IND.2	Exemplo de teste independente
Classe AVA – Declaração de Vulnerabilidade	AVA_MSU.1	Exame dos guias
	AVA_SOF.1	Poder (*strength*) da avaliação da função de segurança do TOE
	AVA_VLA.1	Análise de vulnerabilidades feita pelo desenvolvedor

Tabela 6.4 – Requisitos de Garantia do TOE conforme LSPP.

Cada uma das classes provê medidas de garantia específicas, por exemplo, a classe ATE define requisitos de testes que demonstram que as funções de segurança satisfazem os requisitos funcionais de segurança do TOE. Define testes de cobertura, de profundidade, testes funcionais e independentes. A classe AVA define requisitos direcionados para a identificação e exploração das vulnerabilidades. Especificamente, ela trata das vulnerabilidades introduzidas na construção, operação, uso mal intencionado ou configuração incorreta do TOE.

6.3.5.2.1 Medidas de Garantia do TOE

O TOE *JaCoWeb* satisfaz os requisitos de garantia de segurança especificados em LSPP (NSA, 1999). Essa seção identifica a gerência de configuração, procedimentos de liberação (*delivery*) e operação, procedimentos de desenvolvimento do sistema, documentos guias, medidas de testes e análise de vulnerabilidades aplicadas pela equipe *JaCoWeb* para satisfazer os requisitos EAL3 do CC.

Gerência de configuração

As medidas de gerência de configuração aplicadas pela equipe *JaCoWeb* incluem a atribuição de um único identificador do produto para cada liberação do TOE. Atualmente sua

identificação é Esquema de Autorização *JaCoWeb DiscMand* Versão 1.0. Associado ao identificador do produto está a lista de configuração de *hardware* e *software* que compõe uma única instância do TOE. Essas medidas de gerência de configuração estão sendo registrados dentro do seguinte documento, em desenvolvimento:

- http://www.lcmi.ufsc.br/~merkle/FinalReport.html.

Os **requisitos de garantia satisfeitos no *JaCoWeb*** em relação à gerência de configuração são: ACM_CAP.3 e ACM_SCP.1.

Liberação (*delivery*) e operação

A documentação que descreve a Liberação e Operação do TOE mostra quais componentes são entregues com o *JaCoWeb DiscMand* Versão 1.0, guias para instalação e avisos importantes sobre instalação e configuração do TOE. Essa documentação encontra-se parcialmente apresentada no capítulo 4. Outros elementos estão em desenvolvimento e ficarão disponíveis em:

- http://www.lcmi.ufsc.br/~merkle/FinalReport.html.

Os **requisitos de garantia satisfeitos no *JaCoWeb*** em relação à liberação e operação são: ADO_DEL.1 e ADO_IGS.1.

Desenvolvimento

Os documentos sobre o desenvolvimento do TOE são fornecidos pelos capítulos 4 e 5 que descrevem o Esquema de Autorização proposto, o projeto e a implementação da política discricionária do esquema *JaCoWeb DiscMand* 1.0. A especificação funcional do TOE, o projeto de alto nível e a modelagem das políticas de segurança também são descritos nos capítulos anteriormente citados.

A demonstração da correpondência informal, que compreende a demonstração de que todas as funções de segurança do sistema preenchem os requisitos funcionais estabelecidos no *JaCoWeb-ST*, é justificada pelo próprio modelo de segurança do CORBA, pelos componentes de *software iSaSiLk* (GRAZ, 1999a), *JacORB* (BROSE, 1997), certificados X.509 (ITU-TX509, 1993) e pelo modelo de segurança Java que fornecem as funcionalidades de segurança definidas pelos componentes funcionais do TOE. Os objetos de serviço do CORBAsec utilizados, possuem interfaces e funcionalidades definidas em (OMG, 2000a).

Os **requisitos de garantia satisfeitos no *JaCoWeb*** em relação ao desenvolvimento são: ADV_FSP.1, ADV_HLD.2, ADV_RCR.1 e ADV_SMP.1.

Documentos Guias

Os documentos guias do administrador e do usuário estão disponíveis em

- http://www.lcmi.ufsc.br/~merkle/FinalReport.html.

Esses documentos conduzem o administrador da aplicação desenvolvida sobre os passos a serem seguidos para configurar o TOE. Igualmente os capítulos 4 e 5 desse texto indicam todas as tarefas de responsabilidade do administrador para que o TOE funcione corretamente. O usuário recebe informações sobre a aplicação desenvolvida no capítulo 4 que explica como o TOE opera, quais suas funções e operações disponíveis. Funções relacionadas ao serviço de política *PoliCap* bem como às políticas obrigatórias ainda não encontram-se disponíveis e são objetos de implementações futuras do TOE.

Os **requisitos de garantia satisfeitos no *JaCoWeb*** em relação aos documentos guias são: AGD_ADM.1 e AGD_USR.1.

Suporte ao Ciclo de Vida

A classe de requisitos de garantia que fornecem suporte ao ciclo de vida (classe ALC), define requisitos para garantir a adoção de um modelo de ciclo de vida bem definido para todos os passos do desenvolvimento do TOE, incluindo o correto uso de ferramentas e técnicas e medidas de segurança para proteger o ambiente de desenvolvimento (ISO/IEC 15408-3, 1999).

As medidas adotadas para fornecer suporte ao ciclo de desenvolvimento do TOE foram: estabelecer o uso de uma ferramenta CASE designada *VisualAge for Java* (IBM, 1999) para facilitar a localização e proteção do projeto desenvolvido, estabelecendo uma rotina e centralizando os componentes do *software* e *hardware* utilizados (máquinas *cerf.lcmi.ufsc.br* e *kernighan.lcmi.ufsc.br*). *Backups* diários de modificações foram feitos durante o processo de desenvolvimento. Nenhuma máquina externa ao domínio teve acesso ao TOE em desenvolvimento, neste período.

Os **requisitos de garantia satisfeitos no *JaCoWeb*** em relação ao suporte ao ciclo de vida são: ALC_DVS.1.

Testes

A equipe de desenvolvimento do *JaCoWeb* executou extensivos testes do TOE. Os testes executados incluem ambos testes funcionais, de cobertura (os testes demonstram a correspondência entre documentação e as funções de segurança) e profundidade (os testes são suficientes para

demonstrar que as funções de segurança operam de acordo com o projeto de alto nível do TOE), para garantir que o TOE atende seus objetivos de projeto. A metodologia adotada no processo de testes foi explicada na seção 4.4.5. Os documentos relativos aos testes realizados estão disponíveis no seguinte local:

- http://www.lcmi.ufsc.br/~merkle/FinalReport.html.

Os **requisitos de garantia satisfeitos no *JaCoWeb*** em relação aos testes são: ATE_COV.2, ATE_DPT.1, ATE_FUN.1 e ATE_IND.2.

Declaração de Vulnerabilidades

Como parte do processo de teste e projeto, a equipe *JaCoWeb* executou a Análise de Vulnerabilidades óbvias[38] do *JaCoWeb DiscMand 1.0*. O objetivo dessa análise foi identificar fraquezas óbvias que poderiam ser exploradas para um possível ataque. Neste sentido, o próprio modelo CORBAsec prevê pontos chaves que podem ser aproveitados para tentativas de ataques, e também lista os serviços que o modelo CORBAsec fornece para preveni-los (CHIZMADIA, 2000). Os pontos de ameaças presentes no modelo CORBAsec foram definidos na seção 3.4.9.

Para cada um dos pontos de ataque considerados no modelo CORBAsec, o projeto *JaCoWeb* possui pelo menos um mecanismo de segurança para impedir a ocorrência desses ataques, como pode ser visto na Tabela 6.5. Pontanto, o projeto *JaCoWeb* não apresenta *vulnerabilidades óbvias* no ambiente.

A declaração e análise de vulnerabilidades do projeto *JaCoWeb* está disponível no seguinte local:

- http://www.lcmi.ufsc.br/~merkle/FinalReport.html.

Os **requisitos de garantia satisfeitos no *JaCoWeb*** em relação à declaração de vulnerabilidades são: AVA_MSU.1, AVA_SOF.1 e AVA_VLA.1.

[38] **Vulnerabilidades ou fraquezas óbvias** são aquelas que possibilitam a exploração do ambiente com um mínimo: de entendimento do TOE, de habilidades, de sofisticação técnica e de recursos. Essas vulnerabilidades podem ser sugeridas na descrição do TOE. Vulnerabilidades óbvias incluem aquelas de conhecimento público, ou aquelas conhecidas pelo desenvolvedor ou disponíveis a partir de uma autoridade de avaliação.

Ponto de Ataque	Mecanismos de Segurança do CORBAsec	Mecanismos de Segurança no *JaCoWeb*
1	➢ identificação do usuário, autenticação do usuário; ➢ controle de acesso do usuário.	➢ identificação do usuário, autenticação do usuário; ➢ controle de acesso do usuário.
2	➢ controle de acesso a nível de aplicação; ➢ não-repudiação; ➢ registro de auditoria de segurança; ➢ proteção dos dados.	➢ ----------; ➢ ----------; ➢ ----------; ➢ proteção dos dados.
3	➢ controle de acesso na invocação dos objetos no lado do cliente; ➢ proteção dos dados.	➢ controle de acesso na invocação dos objetos no lado do cliente; ➢ proteção dos dados.
4	➢ autenticação entre cliente e servidor; ➢ cifragem entre cliente e servidor; ➢ controles de delegação; ➢ registro de auditoria de segurança.	➢ autenticação entre cliente e servidor (via SSL); ➢ cifragem entre cliente e servidor (SSL); ➢ ----------; ➢ ----------.
5	➢ cifragem na comunicação; ➢ passagem de IIOP pelos *firewalls*.	➢ cifragem na comunicação; ➢ ----------.
6	➢ controle de acesso na invocação dos objetos no lado do servidor; ➢ domínios de políticas de segurança.	➢ controle de acesso na invocação dos objetos no lado do servidor; ➢ ----------.
7	➢ controle de acesso a nível de aplicação; ➢ não-repudiação; ➢ registro de auditoria de segurança; ➢ proteção dos dados.	➢ ----------; ➢ ----------; ➢ ----------; ➢ proteção dos dados.

Tabela 6.5 – Mecanismos de segurança presentes no *JaCoWeb*.

6.3.6 *Interpretação* (Rationale)

A interpretação dos requisitos funcionais do *JaCoWeb-ST* (Anexo 1) demonstra que o ST é completo e consistente. Como exemplo, o mapeamento entre componentes e alguns objetivos de segurança é representado na Tabela 6.6.

Objetivo de Segurança	Componente Funcional
O_AUTHORIZATION	1.6.1.2 Definição dos Atributos dos Usuários (FIA_ATD.1) 1.6.1.2 Força (*Strength*) dos Dados de Autenticação (FIA_SOS.1) 1.6.1.2 Autenticação (FIA_UAU.1) 1.6.1.2 *Feedback* da Autenticação Protegida (FIA_UAU.7) 1.6.1.2 Identificação (FIA_UID.1) 1.6.1.1 Gerência dos Dados de Autenticação (FMT_MTD.1 (4))
O_DISCRETIONARY_ ACCESS	1.6.1.3 Política de Controle de Acesso Discricionária (FDP_ACC.1) 1.6.1.3 Funções de Controle de Acesso Discricionárias (FDP_ACF.1) 1.6.1.2 Definição dos Atributos dos Usuários (FIA_ATD.1) 1.6.1.2 Ligação (*binding*) entre o usuário e sujeito (FIA_USB.1) 1.6.1.1 Gerência dos Atributos de Segurança do Objeto (FMT_MSA.1) 1.6.1.1 Inicialização dos Atributos Estáticos (FMT_MSA.3) 1.6.1.1 Revogação de Atributos de Objeto (FMT_REV.1 (2))
O_MANDATORY_ ACCESS	1.6.1.3 Exportação de Dados de Usuário Não Rotulados (FDP_ETC.1) 1.6.1.3 Exportação de Dados de Usuário Rotulados (FDP_ETC.2) 1.6.1.3 Política de Controle de Acesso Obrigatória (FDP_IFC.1) 1.6.1.3 Funções de Controle de Acesso Obrigatórias (FDP_IFF.2) 1.6.1.3 Importação de Dados de Usuário Não-Rotulados (FDP_ITC.1) 1.6.1.3 Importação de Dados de Usuário Rotulados (FDP_ITC.2) 1.6.1.2 Ligação (*binding*) entre o usuário e sujeito (FIA_USB.1) 1.6.1.1 Gerência dos Atributos de Segurança do Objeto (FMT_MSA.1) 1.6.1.1 Inicialização dos Atributos Estáticos (FMT_MSA.3)
O.AUDITING	1.6.1.6 Geração de Dados de Auditoria (FAU_GEN.1) 1.6.1.6 Associação da Identidade do Usuário (FAU_GEN.2) 1.6.1.6 Revisão da Auditoria (FAU_SAR.1) 1.6.1.6 Revisão da Auditoria Restrita (FAU_SAR.2) 1.6.1.6 Revisão da Auditoria Selecionada (FAU_SAR.3) 1.6.1.6 Auditoria Seletiva (FAU_SEL.1) 1.6.1.6 Garantias de Disponibilidade de Dados de Auditoria (FAU_STG.1) 1.6.1.6 Ação no caso da possível perda dos Dados de Auditoria (FAU_STG.3) 1.6.1.6 Prevenção da Perda dos Dados de Auditoria (FAU_STG.4) 1.6.1.2 Ligação (*binding*) entre o usuário e sujeito (FIA_USB.1) 1.6.1.1 Gerência da trilha (*trail*) de Auditoria (FMT_MTD.1 (1)) 1.6.1.1 Gerência dos Eventos de Auditoria (FMT_MTD.1 (2)) 1.6.1.4 *Time Stamps* Confiáveis (FPT_STM.1)
O.RESIDUAL_ INFORMATION	1.6.1.3 Proteção às Informações Residuais dos Objetos (FDP_RIP.2)
O.MANAGE	1.6.1.6 Revisão da Auditoria (FAU_SAR.1) 1.6.1.6 Revisão da Auditoria Selecionada (FAU_SAR.3) 1.6.1.6 Auditoria Seletiva (FAU_SEL.1) 1.6.1.6 Ação no caso da possível perda dos Dados de Auditoria (FAU_STG.3) 1.6.1.6 Prevenção da Perda dos Dados de Auditoria (FAU_STG.4) 1.6.1.1 Gerência da trilha (*trail*) de Auditoria (FMT_MTD.1 (1)) 1.6.1.1 Gerência dos Eventos de Auditoria (FMT_MTD.1 (2)) 1.6.1.1 Gerência dos Atributos dos Usuários (FMT_MTD.1 (3)) 1.6.1.1 Gerência dos Dados de Autenticação (FMT_MTD.1 (4)) 1.6.1.1 Revogação de Atributos de Usuário (FMT_REV.1 (1)) 1.6.1.1 Papéis (*roles*) na gerência da segurança (FMT_SMR.1) 1.6.1.5 Acesso a chaves criptográficas (FCS_CKM.3) 1.6.1.5 Destruição de chaves criptográficas (FCS_CKM.4) 1.6.1.5 Operação criptográfica (FCS_COP.1)
O.ENFORCEMENT	1.6.1.4 Teste da Máquina Abstrata (FPT_AMT.1) 1.6.1.4 Mediação de Referência (FPT_RVM.1) 1.6.1.4 Separação de Domínios (FPT_SEP.1) 1.6.1.5 Acesso a chaves criptográficas (FCS_CKM.3) 1.6.1.5 Destruição de chaves criptográficas (FCS_CKM.4) 1.6.1.5 Operação criptográfica (FCS_COP.1)

Tabela 6.6 – Mapeamento entre objetivos de segurança e componentes funcionais (NSA, 1999).

6.4 Relatório Final de Avaliação do *Esquema de Autorização JaCoWeb*

Esta seção apresenta o *Relatório Técnico de Avaliação*, emitido por Carla Merkle Westphall, para o *Esquema de Autorização JaCoWeb DiscMand Versão 1.0*. Esse relatório deve ser a principal fonte de informação usada por um órgão de verificação como o TTAP dos Estados Unidos ou outro órgão nacional competente, para fornecer um Nível de Garantia de Avaliação (*EAL – Evaluation Assurance Level*) EAL 3 para o *Esquema de Autorização JaCoWeb DiscMand Versão 1.0*.

Este relatório apresenta todos os resultados de avaliação, suas justificativas e quaisquer comentários derivados do trabalho executado durante a avaliação. Os requisitos de segurança do TOE referenciados nesse relatório foram extraídos do *JaCoWeb-ST*, Anexo 1. As atividades executadas para conduzir a avaliação foram extraídas dos documentos intitulados:

- *Common Methodology for Information Technology Security Evaluation, Part 2: Evaluation Methodology* (CEMEB, 1999).
- ISO/IEC 15408-1:1999(E) (ISO/IEC 15408-1, 1999), ISO/IEC 15408-2:1999(E) (ISO/IEC 15408-2, 1999), ISO/IEC 15408-3:1999(E) (ISO/IEC 15408-3, 1999). *Information Technology – Security Techniques - Evatuation Criteria for IT Security – Part 1, Part2, Part3*. First edition 01-12-2000.

A equipe de avaliação atribuiu um veredito geral tomando como base (ISO/IEC 15408-1, 1999), concluiu-se pela conformidade do TOE com os requisitos definidos nas partes 2 e 3 do CC (ISO/IEC 15408-2, 1999, ISO/IEC 15408-3, 1999, NSA, 1999). A equipe avaliadora recomendou o nível EAL3 para o TOE na emissão do certificado credenciador (WESTPHALL, FRAGA, *et al.*, 2000). As ações de avaliação seguiram os elementos definidos para o nível EAL 3 do CC.

O nível EAL3 fornece um nível de garantia moderado, provendo, por exemplo, características de segurança requeridas por grande parte do segmento bancário (CEMEB, 1999, HOUSLEY, 2000). As funções de segurança são analisadas usando-se a especificação funcional, documentação de referência e o projeto de alto nível do TOE para entender o comportamento de segurança. A análise é suportada pelos testes independentes de um subconjunto de funções de segurança do TOE, evidências de testes fornecidas pelo desenvolvedor baseadas na especificação funcional e no projeto de alto nível, confirmação seletiva dos resultados de teste do desenvolvedor, análise de 'força' das funções, e evidências de pesquisa por parte do desenvolvedor para encontrar vulnerabilidades óbvias. Além disso, a garantia é obtida pelo uso de controles do ambiente

simulados, da gerência de configuração do TOE e de evidências de procedimentos de liberação segura.

É importante ressaltar que toda a descrição do Relatório Final de Avaliação do Esquema de Autorização *JaCoWeb* segue a metodologia dos Critérios Comuns (ISO/IEC 15408-1, 1999, ISO/IEC 15408-2, 1999, ISO/IEC 15408-3, 1999), com sua linguagem e formatos pré-estabelecidos.

O padrão CC define que as principais entradas para a avaliação são: o conjunto de evidências, o próprio TOE e os *critérios comuns*, com a *metodologia* e o *esquema de avaliação* definidos. O conjunto de evidências é formado a partir de documentos de teste, documentos de projeto, códigos fonte, *hardware*, a avaliação do ST e qualquer outro recurso que possibilite ao avaliador obter informações sobre o TOE. O resultado esperado do processo de avaliação é uma confirmação de que o ST é satisfeito pelo TOE, com um ou mais relatórios documentando a avaliação.

Esta seção está organizada da seguinte maneira: é apresentada uma introdução ao TOE, a descrição de sua arquitetura, os métodos, técnicas e padrões utilizados na avaliação do TOE e os resultados da avaliação. Esses resultados foram obtidos de acordo com a Metodologia de Avaliação da Segurança (CEMEB, 1999) e tomou como base o *JaCoWeb-ST* (Anexo 1) bem como o conjunto de evidências apresentada pela equipe de desenvolvimento do Projeto *JaCoWeb*. Ressaltamos que serão assumidos os passos exatos definidos pela Metodologia de Avaliação da Segurança (CEMEB, 1999) para apresentar cada um dos ítens deste relatório. Toda a divisão e definição dos ítens, que são apresentados na seqüência, são estabelecidos na Metodologia e estão representados na Figura 6.9 (CEMEB, 1999).

6.4.1 Introdução

A introdução ao TOE *JaCoWeb*, seguindo os passos definidos em (CEMEB, 1999), é composta pela sua identificação, pelos conhecimentos relativos ao esquema de avaliação utilizado e pelas referências bibliográficas usadas durante o processo de avaliação. Uma lista de acrônimos usada neste texto é também apresentada.

6.4.1.1 Identificação

A Tabela 6.7 fornece as informações necessárias para identificar e controlar o Relatório Técnico de Avaliação (*ETR – Evaluation Technical Report*), o alvo de segurança (*ST – Security Target*) e o alvo de avaliação (*TOE – Target of Evaluation*), referentes ao Projeto *JaCoweb Security*. Essa tabela também identifica as partes envolvidas com a avaliação.

6.4.1.2 Identificadores de Avaliação (Background)

Para identificar se a avaliação foi conduzida de acordo com os critérios do esquema de avaliação, e verificar que as conclusões do laboratório que realizou os testes são consistentes com os fatos apresentados na avaliação, foi adotado o *Esquema de Avaliação e Validação do Critério Comum* dos Estados Unidos como fundamento teórico para a elaboração desta avaliação (*NIAP Common Criteria Evaluation and Validation Scheme for IT Security* (NIST/NSA, 1999)). Isso porque a experiência americana sobre o CC está bem fundamentada. O processo de certificação desta avaliação fica então sujeito ao NIAP, para a validação do nível de segurança definido – EAL3.

Item	Identificador
Esquema de Avaliação	*Programa de Garantia de Informação Nacional dos Estados Unidos – Esquema de Avaliação e Validação dos Critérios Comuns para Organização, Gerência e Conceitos de Operação da Tecnologia de Informação.* *(National Information Assurance Program NIAP - Common Criteria Evaluation and Validation Scheme for Information Technology Security Organization, Management and Concepts of Operations* (NIST/NSA, 1999)).
Relatório Técnico de Avaliação	A seção 6.4 deste documento intitulada: Relatório Final de Avaliação do Esquema de Autorização *JaCoWeb*.
Protection Profile (PP)	*Labeled Security Protection Profile* (*LSPP*), Version 1.b (NSA, 1999).
Target of Evaluation (TOE)	*Esquema de Autorização JaCoWeb DiscMand Versão 1.0.*
EAL	3
Desenvolvedor	Grupo de Segurança *JaCoWeb*
Responsáveis	Carla Merkle Westphall e Grupo de Segurança *JaCoWeb*
Avaliadores	Carla Merkle Westphall
Certificadores	A serem definidos

Tabela 6.7 – Identificadores de Avaliação.

6.4.1.3 Referências

Os seguintes documentos, citados nas referências bibliográficas, são referenciados ao longo deste texto:

- ISO/IEC 15408-1:1999(E). *Information Technology – Security Techniques - Evatuation Criteria for IT Security - Part 1: Introduction and general model*, First edition 01-12-2000 (ISO/IEC 15408-1, 1999).
- ISO/IEC 15408-2:1999(E). *Information Technology – Security Techniques - Evatuation Criteria for IT Security - Part 2: Security Funcional Requirements*, First edition 01-12-2000 (ISO/IEC 15408-2, 1999).
- ISO/IEC 15408-3:1999(E). *Information Technology – Security Techniques - Evatuation Criteria for IT Security - Part 3: Security Assurance Requirements*, First edition 01-12-2000 (ISO/IEC 15408-3, 1999).

- Common Evaluation Methodology (CEM) Editorial Board (CEMEB). *Common Methodology for Information Technology Security Evaluation, Part 1: Introduction and general model*, CEM-97/017, Draft Version 0.6, 1997 (CEMEB, 1997).
- Common Evaluation Methodology (CEM) Editorial Board (CEMEB). *Common Methodology for Information Technology Security Evaluation, Part 2: Evaluation Methodology*, CEM-99/045, Version 1.0, August 1999 (CEMEB, 1999).
- Information Systems Security Organization. *Labeled Security Protection Profile*. NSA – National Security Agency, Version 1.b, 8 October 1999 (NSA, 1999).
- NIST and NSA. *Common Criteria Evaluation and Validation Scheme for Information Technology Security Organization, Management and Concepts of Operations*, Scheme Publication #1, Version 2.0, May 1999 (NIST/NSA, 1999).
- Common Criteria Project. *Arrangement on the Mutual Recognition of Common Criteria Certificates in the field of IT Security*. 5 October 1998 (NIAP/NIST, 1998).

6.4.1.4 Lista de acrônimos/Glossário de termos

Os seguintes acrônimos são usados neste texto.

CC	Critérios Comuns
CEM	Metodologia de Avaliação Comum
EAL	Nível de Garantia de Avaliação
ETR	Relatório Técnico de Avaliação
PP	*Profile* (perfil) de Proteção
ST	Alvo de Segurança
SAR	Requisito de Garantia de Segurança
SFR	Requisito Funcional de Segurança
TOE	Alvo de Avaliação

6.4.2 Descrição da Arquitetura do TOE

6.4.2.1 Identificação do TOE

O Esquema de Autorização *JaCoWeb DiscMand* Versão 1.0, referenciado aqui como o alvo de avaliação ou TOE (*Target of Evaluation*), consiste dos componentes de *software* e *hardware* descritos na Tabela 6.8.

Componentes	Itens
Software	Pacote *JaCoWebSecurity* Versão 1.0
	Esquema de Autorização *JaCoWeb DiscMand* Versão 1.0 (*applets* de aplicação e servidores de aplicação que usam o pacote *JaCoWebSecurity*)
	JacORB v. 1.0 beta 13
	IAIK-JCE 2.5
	Netscape Communicator 4.5
	Java JDK 1.2.1 da Sun
	Microsoft Windows 95 e FreeBSD 2.2.8
	Apache Web Server 1.3.9
Hardware	2 computadores Intel P133 64Mb em rede
	30 Mb espaço em disco rígido (pelo menos)

Tabela 6.8 – Componentes de *Software/Hardware*.

6.4.2.2 Configuração Avaliada

A configuração avaliada do TOE *JaCoWeb* é formada por uma parte do TOE implementada no ambiente real, como os *applets* de aplicação, servidores de aplicação e o pacote *JaCoWebSecurity*. Considera-se implementadas as políticas discricionárias *do Esquema de Autorização JaCoWeb DiscMand* (implementação descrita no capítulo 4). O serviço de política *PoliCap* tem sua implementação simplificada e, portanto, é avaliado em relação ao esquema definido. As políticas obrigatórias também são avaliadas com base no esquema de autorização definido no capítulo 5.

A Figura 3.16 ilustra a configuração avaliada, representada pelos objetos de serviço do modelo CORBA de segurança e outros objetos presentes no Esquema de Autorização *JaCoWeb DiscMand*.

6.4.2.3 Overview do Sistema

O TOE *JaCoWeb* foi projetado para prover segurança em um ambiente de programação distribuída, através da integração do modelos de segurança CORBA com os modelos de segurança Java e Web, para definir e aplicar políticas de segurança em redes de larga escala como a Internet (WESTPHALL e FRAGA, 1999a, WESTPHALL, FRAGA, *et al.*, 2000).

Os clientes são representados por *applets* Java e os servidores de aplicação por objetos CORBA. Os clientes, a partir dos seus *browsers*, fazem requisições aos objetos servidores, e usam o Esquema de Autorização *JaCoWeb DiscMand* Versão 1.0 de forma transparente, para prover segurança em suas aplicações.

O *Esquema de Autorização JaCoWeb DiscMand* Versão 1.0 é composto pelos seguintes módulos:

- Um *Serviço de política* (*PoliCap*), desenvolvido tendo como suporte a ferramenta JacORB v. 1.0 beta 13 (BROSE, 1997). Este serviço fornece interfaces para o desenvolvimento de aplicações que utilizam políticas de controle de acesso discricionárias e obrigatórias no contexto CORBAsec (conforme definido nos capítulos 4 e 5).
- Um *applet de aplicação* desenvolvido em Java JDK 1.2.1, assinado, que realiza o acesso a um servidor de aplicação CORBA.
- Um servidor de aplicação CORBA, desenvolvido tomando como base o JacORB v. 1.0 beta 13 (BROSE, 1997), que fornece serviços aos clientes representados pelos *applets* Java.
- Um pacote (conjunto de classes) *JaCoWebSecurity*, desenvolvido em Java e CORBA, que implementa, seguindo o nível de segurança *SecurityLevel2*, as interfaces e objetos padronizados do CORBAsec.

A política de segurança, presente no serviço de política *PoliCap*, é definida como global em um sistema distribuído em termos de atributos de privilégio dos principais e atributos de controle dos recursos do sistema, conforme descrito no capítulo 4. Uma vez definida através das interfaces administrativas do *PoliCap*, a política de segurança pode ser usada por aplicações no sentido de prover segurança nas interações entre *applets* e servidores de aplicação, em um ambiente de programação distribuída de larga escala.

6.4.2.4 Escopo e Limites Físico e Lógico

A configuração do TOE consiste de pelo menos dois computadores Intel P133 64Mb em rede, executando o Pacote *JaCoWebSecurity* Versão 1.0, o *browser* Netscape Communicator 4.5 (apenas no computador cliente), os sistemas operacionais Microsoft Windows 95 e FreeBSD 2.2.8, o servidor Web *Apache Web Server* 1.3.9, o *Applet* cliente de aplicação, o Objeto servidor de aplicação, e Aplicações CORBA administrativas para gerenciar o serviço de política *PoliCap* e os objetos de auditoria. A Figura 6.11 representa o escopo lógico do TOE.

As características do TOE são descritas no Anexo 1, item 1.2.2.

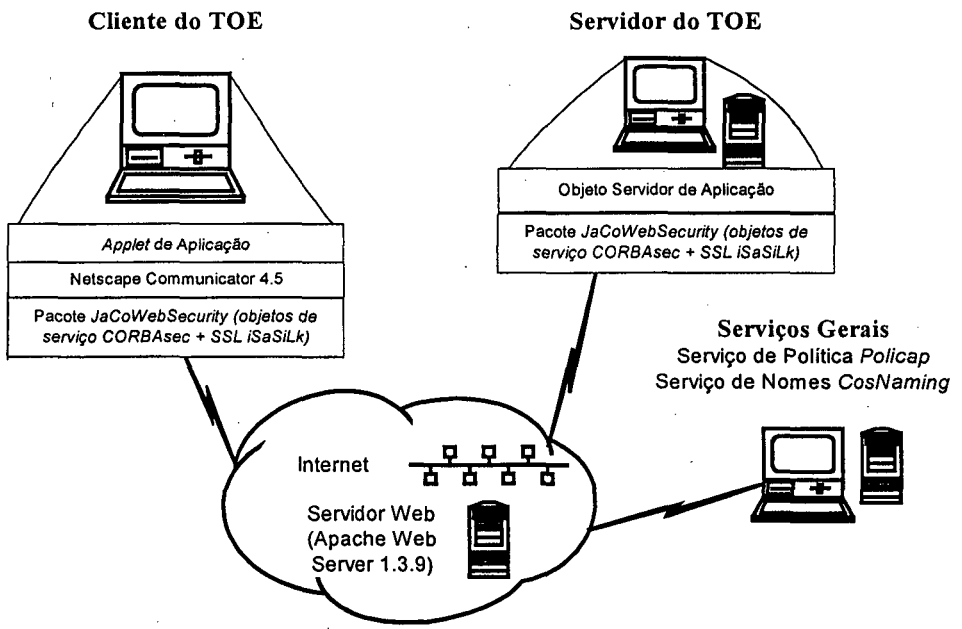


Figura 6.11 – Escopo Lógico do TOE.

6.4.3 Avaliação

6.4.3.1 Métodos, Técnicas e Padrões de Avaliação

O nível EAL3 fornece um nível de garantia moderado (CEMEB, 1999, HOUSLEY, 2000). As funções de segurança são analisadas usando-se a especificação funcional, documentação de referência e o projeto de alto nível do TOE, para entender o comportamento de segurança. A análise é suportada pelos testes independentes de um subconjunto de funções de segurança do TOE, evidências de testes fornecidas pelo desenvolvedor, baseadas na especificação funcional e no projeto de alto nível, confirmação seletiva dos resultados de teste do desenvolvedor, análise de 'força' das funções, e evidências de pesquisa por parte do desenvolvedor para encontrar vulnerabilidades óbvias. Além disso, a garantia é obtida pelo uso dos ambientes de desenvolvimento, da gerência de configuração do TOE, e de evidências de procedimentos de liberação segura.

Os *elementos de ação do avaliador* constituem as ações a serem executadas por um avaliador a fim de avaliar um TOE (CEMEB, 1999). Os elementos de ação do avaliador, documentados em (CEMEB, 1999) para os componentes de garantia do nível EAL3, foram a base da abordagem para a avaliação do TOE. Além disso, o capítulo 7 do documento (CEMEB, 1999) foi usado para definir as ações específicas do avaliador para conduzir a avaliação.

Para gerenciar o esforço de avaliação e para documentar o progresso e as conclusões obtidas, o time de avaliação desenvolveu relatórios de cada grupo de tarefas para cada família de garantia, conforme listado na Tabela 6.9. Cada um dos grupos de tarefas é desenvolvido no item 6.4.4.

Grupo de Tarefas	Componente de Garantia
Alvo de Segurança (*Security Target*)	ASE
Gerência de Configuração	ACM_CAP.3
	ACM_SCP.1
Liberação (*delivery*) e operação	ADO_DEL.1
	ADO_IGS.1
Desenvolvimento	ADV_FSP.1
	ADV_HLD.2
	ADV_RCR.1
	ADV_SMP.1
Documentos Guias	AGD_ADM.1
	AGD_USR.1
Suporte ao Ciclo de Vida	ALC_DVS.1
Testes	ATE_COV.2
	ATE_DPT.1
	ATE_FUN.1
	ATE_IND.2
Declaração de Vulnerabilidades	AVA_MSU.1
	AVA_SOF.1
	AVA_VLA.1

Tabela 6.9 – Grupo de Tarefas para cada componente de garantia.

6.4.3.2 Evidências para a Avaliação

Alguns dos elementos utilizados durante o processo de avaliação são documentos gerados durante o processo de desenvolvimento do TOE, que fazem parte do conjunto de evidências do TOE. A Tabela 6.10 apresenta uma lista de evidências fornecidas para a avaliação, todas acessadas a partir da URL http://www.lcmi.ufsc.br/~merkle/FinalReport.html.

Identificador	Título
[JCW1_CM]	*JaCoWeb*-Gerência de Configuração Versão 1.0
[JCW1_DEL]	*JaCoWeb*-Liberação e Operação Segura Versão 1.0
[JCW1_DOC]	*JaCoWeb*-Guia do Usuário e do Administrador
[JCW1_TST]	*JaCoWeb*-Testes
[JCW1_VUL]	*JaCoWeb*-Declaração de Vulnerabilidades

Tabela 6.10 – Referências para a Avaliação.

6.4.4 Resultados da Avaliação

Esta seção apresenta as conclusões e os resultados da avaliação, onde é feita uma identificação do *veredito* relativo a cada componente de requisito de garantia que constitui uma atividade da avaliação de nível EAL3 do TOE. A Tabela 6.11 fornece uma lista de atividades, associadas com os componentes de garantia, e os elementos de ação correspondentes para uma avaliação de nível EAL3 do TOE. Um *veredito* para um componente de garantia é determinado pelo resultado de vereditos atribuídos aos elementos de ação do avaliador[39]. Três vereditos mutuamente exclusivos são definidos no CC (CEMEB, 1999, ISO/IEC 15408-3, 1999):

- *PASS* (correto), se o avaliador completa com sucesso o elemento de ação do avaliador. As condições para completar um elemento de ação do avaliador com sucesso são definidas pelos grupos de tarefas da ação relacionadas em (CEMEB, 1999).
- *INCONCLUSIVE* (não conclusivo), se o avaliador não concluiu uma ou mais unidades da ação para o elemento de ação do avaliador.
- *FAIL* (falho), se o avaliador completa com insucesso um elemento de ação do avaliador.

A seção 6.4.5 apresenta *o veredito geral* do TOE determinado pela equipe de avaliação conforme definido no capítulo 5 de (ISO/IEC 15408-1, 1999), e fundamentado pelas atribuições de vereditos apresentadas nessa seção.

[39] Elementos de Ação do Avaliador - já foi definido na seção 6.4.3.1.

Grupo de Tarefas	Componente de Garantia	Elementos de Ação do Avaliador
1. Alvo de Segurança (*Security Target*)	ASE_DES.1	ASE_DES.1.1E, ASE_DES.1.2E, ASE_DES.1.3E
	ASE_ENV.1	ASE_ENV.1.1E, ASE_ENV.1.2.E
	ASE_INT.1	ASE_INT.1.1E, ASE_INT.1.2E, ASE_INT.1.3E
	ASE_OBJ.1	ASE_OBJ.1.1E, ASE_OBJ.1.2E
	ASE_PPC.1	ASE_PPC.1.1E, ASE_PPC.1.2E
	ASE_REQ.1	ASE_REQ.1.1E, ASE_REQ.1.2E
	ASE_SRE.1	ASE_SRE.1.1E, ASE_SRE.1.2E
	ASE_TSS.1	ASE_TSS.1.1E, ASE_TSS.1.2E
2. Gerência de Configuração	ACM_CAP.3	ACM_CAP.3.1E
	ACM_SCP.1	ACM_SCP.1.1E
3. Liberação (*delivery*) e operação	ADO_DEL.1	ADO_DEL.1.1E, ADO_DEL.1.2D
	ADO_IGS.1	ADO_IGS.1.1E, ADO_IGS.1.2E
4. Desenvolvimento	ADV_FSP.1	ADV_FSP.1.1E, ADV_FSP.1.2E
	ADV_HLD.2	ADV_HLD.2.1E, ADV_HLD.2.2E
	ADV_RCR.1	ADV_RCR.1.1E
	ADV_SMP.1	ADV_SMP.1.1E
5. Documentos Guias	AGD_ADM.1	AGD_ADM.1.1E
	AGD_USR.1	AGD_USR.1.1E
6. Suporte ao Ciclo de Vida	ALC_DVS.1	ALC_DVS.1.1E, ALC_DVS.1.2E
7. Testes	ATE_COV.2	ATE_COV.2.1E
	ATE_DPT.1	ATE_DPT.1.1E
	ATE_FUN.1	ATE_FUN.1.1E
	ATE_IND.2	ATE_IND.2.1E, ATE_IND.2.2E, ATE_IND.2.3E
8. Declaração de Vulnerabilidades	AVA_MSU.1	AVA_MSU.1.1E, AVA_MSU.1.2E, AVA_MSU.1.3E
	AVA_SOF.1	AVA_SOF.1.1E, AVA_SOF.1.2E
	AVA_VLA.1	AVA_VLA.1.1E, AVA_VLA.1.2E

Tabela 6.11 – Atividades de Avaliação, Componentes de Garantia e Elementos de Ação.

6.4.4.1 *Avaliação do* JaCoWeb-ST

Na Tabela 6.11 o primeiro grupo de ações (Alvo de Segurança) se restringe exclusivamente à avaliação do ST. No Anexo 2, cada componente destes é verificado no contexto do *JaCoWeb-ST*, que representa uma das entradas para a avaliação do TOE.

Na seqüência, as outras ações da Tabela 6.11 são aplicadas no sentido de avaliar o TOE *JaCoWeb* EAL3, em conformidade com o *LSPP* (NSA, 1999).

6.4.4.2 Gerência de Configuração – ACM_CAP.3 e ACM_SCP.1

Os objetivos dessa atividade são determinar se o Esquema de Autorização *JaCoWeb DiscMand* Versão 1.0 identificou claramente o TOE e seus ítens de configuração associados. Essa atividade concretiza os elementos de ação do avaliador identificados no segundo grupo de tarefas da Tabela 6.11.

ACM_CAP.3 – Capacidades da Gerência de Configuração

ACM_CAP.3 Veredito:

A equipe de avaliação concluiu que o TOE atendeu os requisitos de garantia do ACM_CAP.3 pois o elemento de ação do avaliador ACM_CAP.3.1E foi completado com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

ACM_CAP.3.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ACM_CAP.3.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ACM_CAP.3.1E Rationale:

A equipe de avaliação verificou (*checked*) e examinou (*examined*) os documentos [JCW1_CM] e [JCW1_DEL]. O avaliador examinou as evidências e considerou que elas descrevem os procedimentos relacionados à gerência de configuração (CM – *Configuration Management*). O avaliador examinou o software do TOE e a documentação [JCW1_DEL] para determinar que o TOE fornece um único rótulo de identificação pelo uso de um número de versão. O avaliador considerou que as referências do TOE estão consistentes. O avaliador avaliou o [JCW1_CM] para determinar que uma lista de configuração é fornecida e que a lista descreve e identifica de forma única os ítens de configuração do TOE. O avaliador examinou o [JCW1_CM] e determinou que são descritos procedimentos para manter a integridade dos ítens de configuração do TOE, procedimentos que atuam como plano de gerência de configuração (que especifica como os ítens de configuração são mantidos e acessados de forma autorizada pelo sistema). Como resultado dessas atividades, o avaliador determinou que todos os requisitos para essa atividade foram satisfeitos.

ACM_SCP.1 – Escopo da Gerência de Configuração

ACM_SCP.1 Veredito:

A equipe de avaliação concluiu que o TOE atende os requisitos de garantia do ACM_SCP.1 do ST pois o elemento de ação do avaliador ACM_SCP.1.1E foi completado com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

ACM_SCP.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ACM_SCP.1.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ACM_SCP.1.1E Rationale:

A equipe de avaliação verificou (*checked*) e examinou (*examined*) o documento [JCW1_CM]. O avaliador examinou as evidências e considerou que elas definem toda a documentação requerida para atingir o nível de garantia definido, toda a documentação de projeto (projeto de alto nível), fazem referência ao *software* testado e ao *hardware* utilizado, bem como a forma de obter o estado de cada item de configuração ao longo do ciclo de vida do sistema. Como resultado dessa atividade, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

6.4.4.3 Liberação (delivery) e operação –ADO_DEL.1 e ADO_IGS.1

Os objetivos dessa atividade consistem em:

- determinar se os documentos de entrega descrevem todos os procedimentos usados para manter a integridade no momento da distribuição do TOE ao *site* do usuário e,
- determinar se os procedimentos para a instalação, geração e inicialização do TOE foram documentados e resultam em uma configuração segura.

Essa atividade concretiza os elementos de ação do avaliador identificados no terceiro grupo de tarefas da Tabela 6.11.

ADO_DEL.1 – Procedimentos de Liberação

ADO_DEL.1 Veredito:

A equipe de avaliação concluiu que o TOE atende os requisitos de garantia do ADO_DEL.1 pois os elementos de ações do avaliador ADO_DEL.1.1E e ADO_DEL.1.2E foram completados com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

ADO_DEL.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ADO_DEL.1.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ADO_DEL.1.1E Rationale:

A equipe de avaliação revisou o documento [JCW1_DEL]. Os procedimentos de liberação representados em [JCW1_DEL] refletem e melhoram os processos e procedimentos seguros de liberação atuais da equipe *JaCoWeb*. O avaliador determinou que todos os requisitos para essa atividade foram satisfeitos.

ADO_DEL.1.2D Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ADO_DEL.1.2E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ADO_DEL.1.2D Rationale:

A equipe de avaliação revisou o documento [JCW1_DEL]. Embora o TOE não tenha sido entregue a consumidores, por ser fonte de pesquisa, assumiu-se que os procedimentos de liberação utilizados serão os procedimentos representados em [JCW1_DEL]. O avaliador determinou que os requisitos para essa atividade foram satisfeitos.

ADO_IGS.1 – Procedimentos de Instalação, Geração e Inicialização

ADO_IGS.1 Veredito:

A equipe de avaliação concluiu que o TOE atende os requisitos de garantia do ADO_IGS.1 pois os elementos de ações do avaliador ADO_IGS.1.1E e ADO_IGS.1.2E foram completados com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

ADO_IGS.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ADO_IGS.1.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ADO_IGS.1.1E Rationale:

A equipe de avaliação revisou o documento [JCW1_DEL]. A informação fornecida em [JCW1_DEL] foi comparada com a funcionalidade descrita na seção 1.5.1do Anexo 1 a fim de garantir a completitude do documento [JCW1_DEL]. Além disso, [JCW1_DEL] foi examinado para a verificação da consistência com o Guia do Administrador disponível em [JCW1_DOC] a fim de se ter subsídios para completar a tarefa AGD_ADM. Como resultado dessa atividade, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

ADO_IGS.1.2D Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ADO_IGS.1.2D. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ADO_IGS.1.2D Rationale:

A equipe de avaliação revisou o documento [JCW1_DEL]. A informação fornecida em [JCW1_DEL] foi usada para instalar de forma real o TOE em teste e, dessa forma, verificou-se que os passos necessários para a instalação, geração e inicialização estão descritos corretamente. Como resultado dessa atividade, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

6.4.4.4 Desenvolvimento – ADV_FSP.1, ADV_HLD.2, ADV_RCR.1 e ADV_SMP.1

Os objetivos dessa atividade consistem em:

- determinar se o Esquema de Autorização *JaCoWeb DiscMand* 1.0 fornece uma descrição adequada das funções de segurança do TOE e se as funções de segurança fornecidas pelo TOE são suficientes para satisfazer os requisitos funcionais do ST;
- determinar se o projeto de alto nível é suficiente para satisfazer os requisitos funcionais do ST, se fornece uma descrição das TSF's em termos de unidades de estrutura principais com coerência funcional, e se representa a concretização da especificação funcional;
- determinar se o Esquema de Autorização *JaCoWeb DiscMand* 1.0 implementou de forma correta e completa os requisitos do ST e da especificação funcional no projeto de alto nível.

Essa atividade concretiza os elementos de ação do avaliador identificados no quarto grupo de tarefas da Tabela 6.11.

ADV_FSP.1 – Especificação Funcional Informal

ADV_FSP.1 Veredito:

A equipe de avaliação concluiu que o TOE atende os requisitos de garantia do ADV_FSP.1 pois os elementos de ações do avaliador ADV_FSP.1.1E e ADV_FSP.1.2E foram completados com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

ADV_FSP.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ADV_FSP.1.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ADV_FSP.1.1E Rationale:

A equipe de avaliação verificou e examinou o Anexo 1 (*JaCoWeb-ST*), os capítulos 4 e 5 que descrevem o Esquema de Autorização proposto, o projeto e a implementação da política discricionária do esquema *JaCoWeb DiscMand* 1.0, para verificar a existência da especificação do TOE, do projeto de alto nível e da modelagem das políticas de segurança. A especificação funcional foi examinada e considerou-se que contém todo o texto explanatório informal necessário. Os capítulos 4 e 5 foram examinados e considerou-se que eles identificam as TSF's e interfaces externas do TOE e descrevem suas características de segurança usando um estilo informal. O avaliador interpretou textos e figuras em estilo informal. O avaliador definiu, considerando as tarefas do componente ADV_FSP da Metodologia de Avaliação (CEMEB, 1999), que é fornecida uma descrição consistente do TOE, uma descrição completa e correta das interfaces visíveis externamente, uma descrição completa das funções de segurança e uma descrição completa e detalhada do TOE.

A especificação funcional presente nos documentos acima citados, bem como os documentos presentes em [JCW1_DOC] foram examinados e considerou-se que identificam todas as interfaces das funções de segurança do TOE. As interfaces externas ao TOE, o *applet* de aplicação, visível ao usuário, e a aplicação administrativa usada para administrar a política discricionária no sistema, representam as interfaces externas de acesso do TOE. O avaliador consultou as referências identificadas para garantir que cada resposta provida pelas interfaces, exceções e mensagens de erro foram adequada e corretamente especificadas. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

ADV_FSP.1.2E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ADV_FSP.1.2E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ADV_FSP.1.2E Rationale:

A equipe de avaliação examinou o Anexo 1 (*JaCoWeb-ST*), os capítulos 4 e 5 que descrevem o Esquema de Autorização proposto, o projeto e a implementação da política discricionária do esquema *JaCoWeb DiscMand* 1.0, e considerou-se que apresentam evidências de uma correta geração de instâncias dos requisitos funcionais de segurança do TOE. O avaliador verificou que a especificação funcional é uma instância precisa dos requisitos funcionais de segurança do TOE pela construção de uma tabela onde os requisitos funcionais de segurança do TOE foram mapeados às funções de segurança e descrições ou referências de interfaces. As referências foram verificadas de forma que os requisitos foram totalmente satisfeitos.

Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

ADV_HLD.2 – Descrição do Projeto de Alto Nível

ADV_HLD.2 Veredito:

A equipe de avaliação concluiu que o TOE atende os requisitos de garantia do ADV_HLD.2 pois os elementos de ações do avaliador ADV_HLD.2.1E e ADV_HLD.2.2E foram completados com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

ADV_HLD.2.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ADV_HLD.2.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ADV_HLD.2.1E Rationale:

A equipe de avaliação verificou e examinou o Anexo 1 (*JaCoWeb-ST*), os capítulos 4 e 5 que descrevem o Esquema de Autorização proposto, o projeto e a implementação da política discricionária do esquema *JaCoWeb DiscMand* 1.0. Essas evidências descrevem a especificação funcional e o projeto de alto nível do TOE e, considerou-se que contêm todo o texto explanatório informal necessário, descrevendo as características de segurança do TOE, suas interfaces de acesso externas (o *serviço de política PoliCap*; *applets de aplicação*; servidores de aplicação CORBA e o pacote *JaCoWebSecurity*), seus subsistemas e a funcionalidade de segurança de cada subsistema.

O projeto de alto nível descreve o comportamento funcional das TSF's nos capítulos 4 e 5, com a descrição do *Serviço de Política PoliCap* e do *Esquema de Autorização JaCoWeb* nos seus aspectos discricionários e obrigatórios. O projeto e implementação dos *applets de aplicação*; servidores de aplicação CORBA e o pacote *JaCoWebSecurity* é descrito no capítulo 4. Todo o projeto de alto nível identifica o *software* e *hardware* necessário para a implementação das TSF's, identificando os mecanismos de segurança subjacentes necessários e as interfaces visíveis externamente pelo usuário, objetos servidores e administrador do sistema. A descrição do projeto de alto nível separa de forma bastante clara toda a funcionalidade do TOE, identificando objetos que fornecem a política de autorização do sistema. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

ADV_HLD.2.2E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ADV_HLD.2.2E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ADV_HLD.2.2E Rationale:

A equipe de avaliação verificou e examinou o Anexo 1 (*JaCoWeb-ST*), os capítulos 4 e 5 que descrevem o Esquema de Autorização proposto, o projeto e a implementação da política discricionária do esquema *JaCoWeb DiscMand* 1.0. Essas evidências descrevem a especificação funcional e o projeto de alto nível do TOE e, considerou-se que determinam que o projeto de alto nível é a concretização correta, precisa e completa dos requisitos funcionais de segurança do TOE. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

ADV_RCR.1 – Demonstração de Correspondência Informal

ADV_RCR.1 Veredito:

A equipe de avaliação concluiu que o TOE atende os requisitos de garantia do ADV_RCR.1 pois o elemento de ação do avaliador ADV_RCR.1.1E foi completado com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

ADV_RCR.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ADV_RCR.1.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ADV_RCR.1.1E Rationale:

A equipe de avaliação verificou e examinou o Anexo 1 (*JaCoWeb-ST*) e os capítulos 4 e 5 que descrevem o projeto de alto nível do TOE. A análise de correspondência entre a especificação sumária do TOE e a especificação funcional é traçada automaticamente pelo uso do modelo de segurança CORBA, cuja funcionalidade é definida em (OMG, 2000a), e possui objetos de serviços com interfaces padronizadas e definidas em linguagem IDL. Outros componentes utilizados foram o *software iSaSiLk* (GRAZ, 1999a), *JacORB* (BROSE, 1997), certificados X.509 (ITU-TX509, 1993) e o modelo de segurança Java que fornecem as funcionalidades de segurança definidas pelos componentes funcionais do TOE. O avaliador executou essa tarefa considerando a metodologia definida em (CEMEB, 1999) e a unidade de trabalho ADV_FSP.1. O avaliador verificou o mapeamento dos requisitos funcionais e das funções de segurança e descrições de interfaces presentes no TOE. O avaliador verificou a completitude e correção da correspondência entre a especificação sumária e a especificação funcional do TOE, considerando a especificação funcional como uma correta e completa representação das funções de segurança do TOE.

A correspondência entre a especificação funcional e o projeto de alto nível também foi considerada correta e completa representação da especificação funcional. O avaliador verificou, considerando a unidade de trabalho ADV_HLD.2, a correção e completitude dessa correspondência verificando cada SFR e as interfaces externas e internas que implementam cada um dos requisitos. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

ADV_SMP.1 – Modelagem da Política de Segurança

ADV_SMP.1 Veredito:

A equipe de avaliação concluiu que o TOE atende os requisitos de garantia do ADV_SMP.1 pois o elemento de ação do avaliador ADV_SMP.1.1E foi completado com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

ADV_SMP.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ADV_SMP.1.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ADV_SMP.1.1E Rationale:

A equipe de avaliação verificou e examinou o Anexo 1 (*JaCoWeb-ST*), os capítulos 4 e 5 que descrevem o Esquema de Autorização proposto, o projeto e a implementação da política discricionária do esquema *JaCoWeb DiscMand* 1.0. O avaliador examinou as evidências e

considerou que as mesmas contêm todo o texto explanatório informal necessário. As políticas de segurança definidas no Anexo 1 (*JaCoWeb-ST*) estão explicitamente incluídas no modelo proposto e são representadas por requisitos funcionais de segurança específicos. O comportamento de cada um dos tipos de políticas de autorização, discricionária e obrigatória, está claramente descrito no modelo. O avaliador examinou a especificação funcional e determinou que ela identifica todas as funções de segurança que implementam a política de autorização, e considerou consistentes as funções responsáveis pela implementação da política de segurança conforme a sua respectiva especificação funcional. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

6.4.4.5 Documentos Guias – AGD_ADM.1 e AGD_USR.1

Os objetivos dessa atividade consistem em:

- determinar se o guia do administrador descreve como administrar o TOE de forma segura e;
- determinar se o guia do usuário descreve as funções de segurança e interfaces fornecidas pelas TSF's para usuários não administrativos e se o guia fornece instruções para o uso seguro do TOE.

Essa atividade concretiza os elementos de ação do avaliador identificados no quinto grupo de tarefas da Tabela 6.11.

AGD_ADM.1 – Guia do Administrador

AGD_ADM.1 Veredito:

A equipe de avaliação concluiu que o TOE atende os requisitos de garantia do AGD_ADM.1 pois o elemento de ação do avaliador AGD_ADM.1.1E foi completado com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

AGD_ADM.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação AGD_ADM.1.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

AGD_ADM.1.1E Rationale:

A equipe de avaliação verificou e examinou os documentos [JCW1_DOC], o Anexo 1 e a especificação funcional do TOE disponível nos capítulos 4 e 5. O avaliador examinou as evidências e pelo mapeamento de cada um dos requisitos SFR's às funções e interfaces administrativas, determinou que as evidências descrevem as funções e interfaces de segurança administrativas disponíveis baseadas nos requisitos funcionais de segurança definidos no ST. O avaliador examinou as evidências e determinou que as mesmas descrevem como administrar o TOE de forma segura e as funções administrativas relevantes do sistema. O avaliador examinou as evidências e considerou que o guia do administrador está consistente com outras evidências de avaliação como o ST, a especificação funcional, o guia do usuário e procedimentos de instalação. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

AGD_USR.1 – Guia do Usuário

AGD_USR.1 Veredito:

A equipe de avaliação concluiu que o TOE atende os requisitos de garantia do AGD_USR.1 pois o elemento de ação do avaliador AGD_USR.1.1E foi completado com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

AGD_USR.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação AGD_USR.1.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

AGD_USR.1.1E Rationale:

A equipe de avaliação verificou e examinou os documentos [JCW1_DOC], o Anexo 1 e a especificação funcional do TOE disponível nos capítulos 4 e 5. O avaliador examinou as evidências e determinou que as mesmas descrevem as funções e interfaces de segurança para usuários não administrativos do TOE. O avaliador verificou as evidências e determinou que o uso de funções de autenticação foram descritos, determinou ainda que as responsabilidades e o comportamento do usuário foi adequadamente descrito. O avaliador determinou que o guia do usuário está consistente com outras evidências de avaliação como o ST, a especificação funcional, o guia do administrador e procedimentos de instalação. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

6.4.4.6 Suporte ao Ciclo de Vida – ALC_DVS.1

O objetivo dessa atividade consiste em determinar se os controles de segurança do TOE no ambiente de desenvolvimento são adequados para fornecer confidencialidade e integridade durante as fases de projeto e implementação do TOE que é necessária para que a sua operação segura não seja comprometida.

Essa atividade concretiza os elementos de ação do avaliador identificados no sexto grupo de tarefas da Tabela 6.11.

ALC_DVS.1 – Segurança do processo de desenvolvimento

ALC_DVS.1 Veredito:

A equipe de avaliação concluiu que o TOE atende os requisitos de garantia do ALC_DVS.1 pois os elementos de ações do avaliador ALC_DVS.1.1E e ALC_DVS.1.2E foram completados com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

ALC_DVS.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ALC_DVS.1.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ALC_DVS.1.1E Rationale:

A equipe de avaliação verificou e examinou o Anexo 1 (*JaCoWeb-ST*) e o documento [JCW1_CM]. O avaliador verificou que foi utilizado durante o processo de desenvolvimento do TOE, a ferramenta CASE designada *VisualAge for Java* (IBM, 1999) para facilitar a localização e proteção do projeto desenvolvido, estabelecendo uma rotina e centralizando os componentes do *software* e *hardware* utilizados (máquinas *cerf.lcmi.ufsc.br* e *kernighan.lcmi.ufsc.br*). *Backups* diários de modificações foram feitos durante o processo de desenvolvimento. Nenhuma máquina externa ao domínio teve acesso ao TOE em desenvolvimento, neste período. O avaliador verificou que somente os desenvolvedores tiveram acesso ao TOE e aos seus recursos durante o desenvolvimento. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

ALC_DVS.1.2E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ALC_DVS.1.2E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ALC_DVS.1.2E Rationale:

A equipe de avaliação verificou e examinou o Anexo 1 (*JaCoWeb-ST*) e o documento [JCW1_CM]. O avaliador verificou, durante visita realizada enquanto o TOE estava em desenvolvimento, que os procedimentos e medidas de segurança foram aplicados para garantir a integridade e confidencialidade do TOE. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

6.4.4.7 Testes - ATE_COV.2, ATE_DPT.1, ATE_FUN.1 e ATE_IND.2

Os objetivos dessa atividade consistem em:

- determinar se a evidência dos testes de cobertura mostram a correspondência entre os testes identificados na documentação de teste e a especificação funcional;
- determinar se o desenvolvedor testou as funções de segurança do TOE (TSF's) considerando o projeto de alto nível do mesmo;
- determinar se os testes funcionais do *JaCoWeb* demonstram que todas as funções de segurança são executadas conforme especificadas e;
- determinar se o TOE se comporta conforme especificado para se ganhar a confiança nos resultados dos testes do *JaCoWeb* com os testes independentes de um subconjunto das TSF's e pela execução de uma amostra dos testes do desenvolvedor.

Essa atividade concretiza os elementos de ação do avaliador identificados no sétimo grupo de tarefas da Tabela 6.11. Os testes estão documentados em [JCW1_TST].

ATE_COV.2 – Evidência de Cobertura

ATE_COV.2 Veredito:

A equipe de avaliação concluiu que o TOE atende os requisitos de garantia do ATE_COV.2 pois o elemento de ação do avaliador ATE_COV.2.1E foi completado com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

ATE_COV.2.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ATE_COV.2.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ATE_COV.2.1E Rationale:

A equipe de avaliação verificou e examinou o documento [JCW1_TST] e a especificação funcional do TOE disponível nos capítulos 4 e 5. O avaliador examinou e verificou os testes executados pelo desenvolvedor, conforme documentados em *http://www.lcmi.ufsc.br/~merkle/FinalReport.html*, e considerou que esses testes são suficientes para estabelecer que as funções TSF's foram sistematicamente testadas considerando a especificação funcional. As unidades básicas a serem testadas: applet de aplicação do servidor bancário, servidor bancário CORBA de aplicação, aplicação administrativa Java usada para definir as políticas de segurança discricionárias do sistema, objetos de serviço do CORBAsec que executam tarefas relacionadas ao controle de acesso do *JaCoWeb* e objetos de serviço do CORBAsec que executam tarefas relacionadas ao fornecimento de confidencialidade e integridade da requisição do *JaCoWeb* foram testadas. O avaliador considerou que todas as funções de segurança do sistema foram testadas. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

ATE_DPT.1 – Profundidade dos Testes

ATE_DPT.1 Veredito:

A equipe de avaliação concluiu que o TOE atende os requisitos de garantia do ATE_DPT.1 pois o elemento de ação do avaliador ATE_DPT.1.1E foi completado com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

ATE_DPT.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ATE_DPT.1.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ATE_DPT.1.1E Rationale:

A equipe de avaliação verificou e examinou o documento [JCW1_TST] e a especificação funcional do TOE disponível nos capítulos 4 e 5, o projeto de alto nível e a análise de profundidade dos testes elaborada pelo desenvolvedor, bem como a documentação de testes do desenvolvedor (*http://www.lcmi.ufsc.br/~merkle/FinalReport.html*). O avaliador examinou a análise da profundidade de testes elaborada pelo desenvolvedor, verificou os testes executados pelo desenvolvedor, conforme documentados em *http://www.lcmi.ufsc.br/~merkle/FinalReport.html*, e considerou que o desenvolvedor testou as funções TSF's considerando o projeto de alto nível do TOE. O avaliador verificou que existe um mapeamento entre os testes identificados na documentação de testes e o projeto de alto nível e, examinou o plano de testes do desenvolvedor e considerou que a abordagem de

testes trata cada uma das TSF's. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

ATE_FUN.1 – Testes Funcionais

ATE_FUN.1 Veredito:

A equipe de avaliação concluiu que o TOE atende os requisitos de garantia do ATE_FUN.1 pois o elemento de ação do avaliador ATE_FUN.1.1E foi completado com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

ATE_FUN.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ATE_FUN.1.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ATE_FUN.1.1E Rationale:

A equipe de avaliação verificou e examinou [JCW1_TST], comparando esse documento conforme requerido à especificação funcional do TOE disponível nos capítulos 4 e 5, bem como ao Anexo 1 (*JaCoWeb-ST*). A documentação de testes fornecida em [JCW1_TST] contêm plano de teste, procedimentos de teste e resultados esperados e reais dos testes. O documento identifica as funções de segurança a serem testadas, os objetivos de cada teste e fornece instruções adequadas para a repetição de todos os testes. Resultados reais atingidos são consistentes com os resultados esperados. Os resultados esperados foram completados com sucesso e demonstraram de forma adequada que a funcionalidade de segurança foi testada. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

ATE_IND.2 – Testes Independentes - Exemplo

ATE_IND.2 Veredito:

A equipe de avaliação concluiu que o TOE atende os requisitos de garantia do ATE_IND.2 pois os elementos de ações do avaliador ATE_IND.2.1E, ATE_IND.2.2E e ATE_IND.2.3E foram completados com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

ATE_IND.2.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ATE_IND.2.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ATE_IND.2.1E Rationale:

A equipe de avaliação utilizou o mesmo laboratório e ambiente computacional da equipe de desenvolvedores para usufruir da descrição do TOE presente no ST e executou seus testes conforme especificados em [JCW1_TST]. Dessa forma, a equipe de avaliação é capaz de garantir que os recursos fornecidos pelo desenvolvedor são equivalentes aos recursos usados pelo desenvolvedor para testar a funcionalidade das TSF's. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

ATE_IND.2.2E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ATE_IND.2.2E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ATE_IND.2.2E Rationale:

A equipe de avaliação conduziu testes independentes com o objetivo de garantir (a) cobertura de todos os SFRs através da combinação dos testes realizados pelo desenvolvedor e pelo avaliador; e (b) um nível de rigor apropriado de testes para funções de segurança críticas. Depois de examinar o documento [JCW1_TST], o avaliador decidiu realizar os seguintes testes:

- **Applet de aplicação e servidor bancário CORBA de aplicação** – testar valores limites, valores normais e testar grafos de causa-efeito;
- **Aplicação Administrativa Java usada para definir as políticas de segurança discricionárias do sistema** – para definir políticas diferentes das políticas utilizadas pelo desenvolvedor durante os testes;
- **Objetos de Serviço do CORBAsec que executam tarefas relacionadas ao Controle de Acesso do *JaCoWeb*** – para garantir que esses objetos atendem seus requisitos e funcionalidades;
- **Objetos de Serviço do CORBAsec que executam tarefas relacionadas ao fornecimento de confidencialidade e integridade da requisição do *JaCoWeb*** – para garantir que esses objetos forneçam integridade e confidencialidade nas invocações.

Os testes foram executados a partir do *applet* de aplicação e também foram feitos testes com o auxílio da ferramenta *VisualAge for Java* (IBM, 1999), para realizar alguns conjuntos de *traces* dos objetos de serviço e garantir sua correta execução e padronização.

Cada teste foi executado pelo menos três vezes para garantir resultados consistentes do TOE. Resultados reais corresponderam aos resultados esperados. Testes independentes estão documentados em *http://www.lcmi.ufsc.br/~merkle/FinalReport.html*. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

ATE_IND.2.3E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ATE_IND.2.3E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ATE_IND.2.3E Rationale:

A abordagem do desenvolvedor foi testar todas as TSF's do TOE no ambiente de laboratório. Os testes do desenvolvedor foram insuficientes para fornecer subsídios ao avaliador para gerar novos testes no TOE. O desenvolvedor, forneceu tão logo isso foi detectado, novas séries de jogos de testes para a verificação da funcionalidade e cobertura do TOE. Todos os resultados de testes foram atingidos com sucesso. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

6.4.4.8 Declaração de Vulnerabilidades

Os objetivos dessa atividade consistem em:

- determinar a existência de falhas e fraquezas que podem ser exploradas no ambiente do TOE, baseando-se na análise executada pelo desenvolvedor e pelo avaliador, suportada pelos testes do avaliador;

- determinar se os guias do usuário, administrador e de instalação e configuração não possuem procedimentos conflitantes e que levem a estados inseguros do sistema, se procedimentos de segurança foram tratados para todos os modos de operação e se os guias facilitam a prevenção e detecção de estados inseguros do TOE;
- determinar se a força das funções de segurança é suportada por uma análise correta;
- determinar se o TOE, em seu ambiente de operação, possui vulnerabilidades óbvias.

Essa atividade concretiza os elementos de ação do avaliador identificados no oitavo grupo de tarefas da Tabela 6.11.

AVA_MSU.1 – Uso Incorreto

AVA_MSU.1 Veredito:

A equipe de avaliação concluiu que o TOE atende os requisitos de garantia do AVA_MSU.1 pois os elementos de ações do avaliador AVA_MSU.1.1E, AVA_MSU.1.2E e AVA_MSU.1.3E foram completados com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

AVA_MSU.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação AVA_MSU.1.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

AVA_MSU.1.1E Rationale:

A equipe de avaliação verificou e examinou o Anexo 1 (*JaCoWeb-ST*), o documento [JCW1_TST] e a especificação funcional do TOE disponível nos capítulos 4 e 5, o projeto de alto nível, os guias de usuário e administrador [JCW1_DOC] bem como a documentação de testes do desenvolvedor (*http://www.lcmi.ufsc.br/~merkle/FinalReport.html*). O avaliador verificou que os documentos guias são claros, internamente consistentes e completos, verificou que identificam os modos de operação do TOE com suas possíveis exceções durante a execução e descrevem as suposições consideradas sobre o ambiente. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

AVA_MSU.1.2E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação AVA_MSU.1.2E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

AVA_MSU.1.2E Rationale:

A equipe de avaliação verificou e examinou o Anexo 1 (*JaCoWeb-ST*), o documento [JCW1_TST] e a especificação funcional do TOE disponível nos capítulos 4 e 5, o projeto de alto nível, os guias de usuário e administrador [JCW1_DOC] bem como a documentação de testes do desenvolvedor (*http://www.lcmi.ufsc.br/~merkle/FinalReport.html*). O avaliador considerou que as evidências determinam os procedimentos necessários para configurar e instalar o TOE de forma segura. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

AVA_MSU.1.3E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação AVA_MSU.1.3E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

AVA_MSU.1.3E Rationale:

A equipe de avaliação verificou e examinou o Anexo 1 (*JaCoWeb-ST*), o documento [JCW1_TST] e a especificação funcional do TOE disponível nos capítulos 4 e 5, o projeto de alto nível, os guias de usuário e administrador [JCW1_DOC] bem como a documentação de testes do desenvolvedor (*http://www.lcmi.ufsc.br/~merkle/FinalReport.html*). O avaliador considerou que as evidências possuem procedimentos suficientes para que os consumidores possam administrar e usar as funções de segurança do TOE de forma efetiva, detectando estados inseguros. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

AVA_SOF.1 – Força das Funções de Segurança

AVA_SOF.1 Veredito:

A equipe de avaliação concluiu que o TOE atende os requisitos de garantia do AVA_SOF.1 pois os elementos de ações do avaliador AVA_SOF.1.1E e AVA_SOF.1.2E foram completados com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

AVA_SOF.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação AVA_SOF.1.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

AVA_SOF.1.1E Rationale:

A equipe de avaliação verificou e examinou o Anexo 1 (*JaCoWeb-ST*), a especificação funcional do TOE disponível nos capítulos 4 e 5, o projeto de alto nível, os guias de usuário e administrador [JCW1_DOC]. O avaliador verificou que foram utilizados o *iSaSiLk* (GRAZ, 1999a), versão do SSL em Java, durante o estabelecimento da conexão segura usando o CORBA. Além do SSL, o pacote IAIK-JCE (GRAZ, 1999b) implementa os algoritmos criptográficos usados pelo *iSaSiLk*. O algoritmo de chave pública, usado para autenticação mútua e distribuição de chaves , é o RSA de 1024 bits. Para a cifragem dos dados utiliza-se o algoritmo simétrico de cifragem em bloco 3DES_EDE e a função hash de segurança SHA para a computação do MAC (capítulo 4). A assinatura dos *applets*[40] clientes é feita usando recursos da linguagem Java JDK 1.2.1. Nenhuma análise de força das funções de segurança do TOE é feita, visto que os algoritmos utilizados são implementações disponíveis e já documentadas na literatura, demonstrando que a força das funções criptográficas pode ser considerada *SOF-medium* (um nível de força onde a análise mostra que a função fornece proteção adequada contra ataques intencionais ou diretos de invasores com potencial de ataque moderado) (ISO/IEC 15408-1, 1999, SCHNEIER, 1996). Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

[40] Um relatório do processo de assinatura do *applet* cliente está disponível no endereço http://www.lcmi.ufsc.br/jacoweb/documentos.

AVA_SOF.1.2E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação AVA_SOF.1.2E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

AVA_SOF.1.2E Rationale:

A equipe de avaliação verificou e examinou o Anexo 1 (*JaCoWeb-ST*), a especificação funcional do TOE disponível nos capítulos 4 e 5, o projeto de alto nível, os guias de usuário e administrador [JCW1_DOC]. O avaliador determinou que não existem declarações de conformidade a algum nível SOF, visto que são usados algoritmos disponíveis na literatura (SCHNEIER, 1996). Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

AVA_VLA.1 – Análise de Vulnerabilidades

AVA_VLA.1 Veredito:

A equipe de avaliação concluiu que o TOE atende os requisitos de garantia do AVA_VLA.1 pois os elementos de ações do avaliador AVA_VLA.1.1E e AVA_VLA.1.2E foram completados com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

AVA_VLA.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação AVA_VLA.1.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

AVA_VLA.1.1E Rationale:

A equipe de avaliação verificou e examinou o Anexo 1 (*JaCoWeb-ST*), a especificação funcional do TOE disponível nos capítulos 4 e 5, o projeto de alto nível, os guias de usuário e administrador [JCW1_DOC], procedimentos de instalação e geração do TOE, análise de vulnerabilidades disponível no *http://www.lcmi.ufsc.br/~merkle/FinalReport.html* e o TOE disponível para teste. O avaliador examinou a análise de vulnerabilidades do desenvolvedor e considerou que todas as informações relevantes foram consideradas para detectar as vulnerabilidades óbvias e interpretar essas vulnerabilidades dentro do contexto do TOE. O avaliador considerou que a análise de vulnerabilidades do desenvolvedor está consistente com o ST e com os guias. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

AVA_VLA.1.2E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação AVA_VLA.1.2E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

AVA_VLA.1.2E Rationale:

A abordagem da equipe de avaliação para os testes de penetração levou em consideração a análise de vulnerabilidades apresentada em *http://www.lcmi.ufsc.br/~merkle/FinalReport.html*, e as vulnerabilidades incluídas em (CHIZMADIA, 2000). Os testes de penetração foram divididos na seguintes áreas:

- *Análise da lista de vulnerabilidades.* O avaliador verificou e examinou o comportamento do TOE e conduziu os testes para tentar encontrar alguma falha na análise do desenvolvedor.

- *Análise de Trafego usando ferramenta sniffer*[41]. O avaliador usou uma ferramenta de *sniffer* da rede para capturar pacotes transmitidos na comunicação e descobrir alguma informação sensível do sistema.

Depois de conduzir os testes de penetração do TOE, a equipe de avaliação concluiu que o TOE não possui vulnerabilidades óbvias que podem ser exploradas para violar a política de segurança do ambiente. Resultados dos testes de penetração estão documentados em *http://www.lcmi.ufsc.br/~merkle/FinalReport.html*. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

6.4.5 Conclusões e Recomendações da Avaliação do JaCoWeb

O TOE foi avaliado considerando o Anexo 1 (*JaCoWeb-ST*). Os vereditos dos componentes de garantia apresentados aqui recebem o veredito final de avaliação **CORRETO** (***PASS***). Portanto, a equipe de avaliação atribui um veredito geral **CORRETO** (***PASS***) por satisfazer os elementos de ação do avaliador definidos para o nível EAL3. Com base no definido em (ISO/IEC 15408-1, 1999), o TOE foi considerado em conformidade com as partes 2 e 3 do CC (ISO/IEC 15408-2, 1999, ISO/IEC 15408-3, 1999) e também com (NSA, 1999). A equipe de avaliação recomenda o nível EAL3 para a emissão de um certificado a ser emitido para o TOE.

6.4.6 Relatórios de Observação

Sendo opcional, nenhum relatório de observação foi gerado.

6.5 Conclusões do Capítulo

Este capítulo apresentou a avaliação do Esquema de Autorização *JaCoWeb* de acordo com o padrão ISO/IEC 15408 (Critérios Comuns). Os principais conceitos sobre este padrão foram apresentados. A definição do *JaCoWeb-ST* foi baseada no requisitos padronizados pelo *LSPP*, e serviu como base para a avaliação e para a elaboração do relatório final de avaliação do *JaCoWeb*.

O uso destes critérios para a realização de uma avaliação da segurança de um sistema apresenta desvantagens e vantagens. As desvantagens que podem ser citadas são as seguintes:

- Observamos que quanto maior o nível de segurança pretendido, maior também é o custo de uma avaliação.
- Os documentos que definem este padrão, estabelecem como usá-lo e apresentam a sua metodologia de avaliação são extensos; muitos detalhes precisam ser entendidos e considerados durante todo o processo de avaliação.
- A quantidade de documentação gerada durante uma avaliação é volumosa.

[41] Uma ferramenta de *sniffer* possibilita a realização de uma 'escuta' dos pacotes que trafegam na rede de comunicação.

Dentre as vantagens que podemos citar estão as seguintes:

- CC fornece um *framework* amplo e completo para a elaboração de uma avaliação de segurança.
- CC é um excelente guia durante o projeto, implementação e avaliação da segurança de um produto ou sistema de tecnologia de informação.
- CC apresenta formas de expressar requisitos de segurança (tanto funcionais quanto de garantia), que geram especificações precisas.
- Atualmente já existe a ferramenta *CC Toolbox* (http://www.commoncriteria.org) que auxilia a gerar, de forma automatizada, toda a documentação necessária à avaliação.

Conforme mais e mais organizações trabalhem em escala global, a importância de padrões internacionais como os Critérios Comuns, cujos certificados de avaliação são reconhecidos por diferentes países, independente do local onde a avaliação foi realizada, torna-se evidente. A uniformidade alcançada através destes certificados reconhecidos internacionalmente representa um avanço para o estabelecimento e entendimento da segurança dos produtos e sistemas de tecnologias de informação em um mercado mundial (CAPLAN e SANDERS, 1999).

A experiência no uso destes critérios em um processo de avaliação de segurança não tem precedentes na comunidade científica brasileira. Neste sentido, nossa experiência é inovadora.

Capítulo 7

CONCLUSÕES

7.1 Revisão das motivações e objetivos

Na medida em que o interesse pela especificação e pelo gerenciamento de políticas de autorização em sistemas que usam objetos distribuídos de larga escala torna-se evidente e indispensável para o estabelecimento de ambientes de programação seguros, começam a aparecer algumas propostas de arquiteturas de segurança para suprir essas necessidades. O modelo CORBA de segurança proposto pela OMG supre algumas destas necessidades na sua especificação.

Uma grande abertura de oportunidades em pesquisa e desenvolvimento é criada para estabelecer esquemas de autorização, fundamentados em *middlewares*. O modelo CORBAsec fornece um conjunto de abstrações e mecanismos que facilitam a especificação e o gerenciamento de políticas de autorização. Aplicar estas políticas de forma transparente ao usuário final de uma aplicação é sempre desejável nestes casos. Estes esquemas devem implementar mecanismos de controle de acesso e de autenticação, fundamentados em uma variedade de tecnologias de segurança, levando em consideração as políticas de autorização do ambiente definidas sobre modelos de segurança apropriados.

Esta tese se inclui nesse contexto, através da proposição de um Esquema de Autorização para aplicações distribuídas usando abstrações do modelo CORBA de segurança integrado aos suportes de segurança das plataformas Java e Web. Este esquema de autorização foi testado como meio de expressão dos modelos Matriz de Acesso (controle de acesso discricionário), Bell e Lapadula (controle obrigatório) e modelos Baseados em Papéis. Esperamos que os trabalhos realizados tenham demonstrado isso e, de certa forma, tenham respondido às perguntas listadas no capítulo 1 que serviram de desafios nesta tese.

7.2 Visão geral do trabalho

O texto desta tese, inicialmente, efetuou um levantamento sobre a segurança em sistemas distribuídos, discutindo alguns modelos de segurança usados para guiar a definição das políticas de segurança e também, as diversas formas utilizadas para implementar os mecanismos de segurança nestes sistemas distribuídos.

Um suporte de segurança para sistemas distribuídos abertos foi identificado a partir de três COTS: CORBA, Java e Web. Um enfoque especial foi dado na interpretação das especificações do modelo CORBA de segurança.

Foi proposto e implementado um esquema de autorização chamado *JaCoWeb* (http://www.lcmi.ufsc.br/jacoweb), que integrou os suportes de segurança das plataformas CORBA, Java e Web, constituindo uma arquitetura de segurança que conta com mecanismos de autenticação, controle de acesso, controles criptográficos e mecanismos de auditoria. Este esquema de autorização, na implementação de controles discricionários (modelo Matriz de Acesso), identificou dois níveis de controle de acesso. O nível global, representado pelo serviço de política *PoliCap* proposto, concretiza o primeiro nível de controle de acesso do esquema de autorização, preenchendo uma lacuna no gerenciamento de políticas de segurança existente no modelo CORBA de segurança. Este primeiro nível é concretizado através da cooperação entre o serviço de política *PoliCap* e o objeto *AccessDecision* do CORBAsec, no lado do cliente, perfazendo o papel de servidor de autorização do sistema. O segundo nível de controle de acesso é desenvolvido com o uso de *capabilities* propostas para o modelo CORBA de segurança. Este segundo nível é concretizado através da construção de *capabilities* no lado do cliente que são verificadas pelo objeto *AccessDecision* no lado do servidor, e também, pelo modelo de segurança Java, através de seus arquivos de política que identificam e verificam as operações permitidas nos objetos locais.

O esquema de autorização *JaCoWeb* foi usado na implementação de controles obrigatórios. O modelo Bell e Lapadula foi usado como base no sentido de implementar políticas obrigatórias em nosso esquema de autorização. Embora de maneira simplificada, controles de acesso baseados em papéis também foram concretizados em nosso esquema.

Uma avaliação da segurança do esquema de autorização proposto, considerando a implementação simultânea de controles obrigatórios e discricionários, foi feita usando o padrão ISO 15408, também chamado *Critérios Comuns para a Avaliação da Segurança de Tecnologias de Informação*, como base conceitual na realização desta avaliação. Concluiu-se que o esquema proposto possui a classificação de segurança EAL3 definida por este padrão.

7.3 Contribuições e escopo do trabalho

Considerando-se os objetivos iniciais traçados, algumas contribuições desta tese podem ser citadas. As principais provavelmente são:

i) A proposição e implementação de um Esquema de Autorização Discricionário para aplicações de objetos distribuídos. O esquema fornece segurança para aplicações de forma transparente aos usuários, aplicada a nível de ORB, e usa suportes de segurança adequados à ambientes de larga escala como o CORBAsec, Java e Web.

ii) A proposição de um Esquema de Autorização Obrigatório, baseado no modelo Bell e Lapadula, que pode ser usado para assegurar aplicações de objetos distribuídos usando as regras definidas no esquema. Estes controles obrigatórios estendem os controles discricionários já existentes no CORBAsec, visando a implantação de outro tipo de política no ambiente, neste caso, não-discricionária. As regras obrigatórias propostas não provocam bloqueios de requisições já processadas.

iii) A proposição de um serviço de política - o *PoliCap* - que supre as carências existentes no CORBAsec no que tange o gerenciamento de políticas de segurança. Este serviço centraliza os controles discricionários e obrigatórios do domínio, fornecendo apenas as informações necessárias sobre os objetos de política para cada associação segura e cada requisição de método.

iv) A definição das *capabilities* a serem usadas no ambiente CORBAsec. O controle de acesso usando *capabilities* não é padronizado pela OMG e, no nosso esquema, o seu uso determinou uma redução no tráfego de rede em caso de negações de acesso ao objeto servidor e também a concretização de um segundo nível de controle de acesso que é realizado no lado do servidor.

v) O uso de uma metodologia de avaliacão de segurança no esquema de autorização proposto, baseada no padrão ISO 15408. Todos os passos necessários para concretizar esta avaliação, de acordo com os Critérios Comuns, foram desenvolvidos gerando a documentação necessária que se encontra disponível.

A concretização do esquema e a implementação dos diferentes modelos de política, usando aplicações exemplos, ajudou numa melhor compreensão do modelo de segurança CORBA nos seus aspectos dinâmicos. A especificação CORBA de segurança é a mais extensa especificação de COSS da OMG. Consideramos que nossas descrições, se não contribuem para um melhor entendimento destas especificações, pelo menos abrem o debate sobre as mesmas na comunidade brasileira de segurança.

7.4 Perspectivas futuras

Existem diversas possibilidades de continuidade deste trabalho. Algumas propostas de possíveis extensões são:

- Implementação dos objetos definidos no *PoliCap*, onde os objetos locais do esquema de autorização são obtidos a partir de objetos globais. Esta implementação faz parte de uma dissertação de mestrado em andamento.
- Implementação de uma aplicação no sentido de evidenciar as políticas de controle de acesso obrigatórias definidas em nossa proposta. Esta implementação também faz parte de uma dissertação de mestrado em andamento.
- A proposição e implementação de políticas de controle de acesso baseadas em papéis, a serem inseridas no *PoliCap*. Um estudo sobre controle dinâmico de papéis está sendo feito no âmbito de uma dissertação de mestrado em andamento.
- O refinamento e implementação da proposição do *JaCoWeb* em um contexto de larga escala, possivelmente usando o LDAP, conforme definições apresentadas no capítulo 3.
- A submissão da apreciação da avaliação de segurança realizada a um órgão internacional de padronização, a fim de obter um certificado de garantia apropriado.

ANEXO 1 – *JACOWEB-ST*

1.1 Introdução ao *Security Target*

Essa seção introdutória apresenta informações de identificação do *Security Target* (ST) e um resumo da estrutura do ST. Uma breve discussão da metodologia de desenvolvimento também é fornecida.

Um documento ST fornece a base para a avaliação de um produto ou sistema de tecnologia de informação (*IT – Information Technology*). Um ST define principalmente:

- Um conjunto de suposições sobre os aspectos de segurança do ambiente, uma lista de ameaças que o produto pretende conter e regras ou políticas que o produto deve suportar;
- Um conjunto de objetivos de segurança e um conjunto de requisitos de segurança para tratar o problema;
- As funções de segurança da IT fornecidas pelo TOE que satisfazem o conjunto de requisitos.

Um ST de um TOE é a base para a concordância entre desenvolvedores, avaliadores e consumidores sobre as propriedades de segurança do TOE e o seu escopo de avaliação. A estrutura e os conteúdos desse ST seguem os requisitos especificados no CC, Parte 1 – Anexo C (ISO/IEC 15408-1, 1999) e Parte 3 – Capítulo 5 (ISO/IEC 15408-3, 1999).

Os conteúdos das seções do ST, sob certas condições, podem ser idênticas às seções do PP, especificamente, quando o ST:

- Declara (*claims*) conformidade com o PP;
- Não inclui nenhuma operação[42] adicional nos requisitos funcionais de segurança do PP;
- Não estende o PP pela adição de objetivos de segurança e/ou requisitos de segurança.

Sob essas condições, o CC afirma que "a *referência* ao PP é suficiente para definir e justificar os objetivos e requisitos do TOE. A *repetição* dos conteúdos do PP é desnecessário"[43].

A metodologia usada para desenvolver e apresentar esse ST inclui os seguintes passos:

- Os objetivos e requisitos de segurança do PP com os quais o ST declara conformidade e para os quais não foram adicionadas operações a serem executadas são reafirmadas dentro do ST explicitamente;

[42] O CC permite o ajuste controlado dos seus requisitos de segurança funcionais, por meio de quatro operações (designadas, refinamento, seleção, atribuição e iteração; ver CC Parte 2, par. 2.1.4).
[43] CC Parte 1, Anexo C, par. C.2.8, b.

- Se o ST irá executar operações adicionais nos requisitos do PP, o ST reafirma os requisitos do ST, executa as operações e identifica a mudança;
- Se o ST estende o PP pela adição de objetivos de segurança e/ou requisitos de segurança, o ST declara os objetivos e/ou requisitos, faz quaiquer adições necessárias à seção Ambiente de Segurança e documenta as seções apropriadas do *Rationale* (Interpretação).

1.1.1 Identificação do TOE

Essa seção fornece informações necessárias para identificar o ST e o seu TOE, o *Esquema de Autorização JaCoWeb DiscMand*, Versão 1.0, considerando políticas discricionárias e obrigatórias do esquema.

- **Título do ST**: Esquema de Autorização *JaCoWeb DiscMand* Versão 1.0 *Security Target*
- **Versão do ST**: 1.0
- **Identificação do TOE**: Esquema de Autorização *JaCoWeb DiscMand* Versão 1.0
- **Identificação do CC**: Information Technology – Security Techniques - Evatuation Criteria for IT Security ISO 15408
- **Identificação do PP**: *Labeled Security Protection Profile*, Version 1.b, 8 October 1999 (referenciado como LSPP)
- **Avaliação do ST**: Carla Merkle Westphall
- **Palavras-chave**: controle de acesso, controle de acesso discricionário, controle de acesso obrigatório, segurança, *security target*

1.1.2 Convenções, Terminologia e Acrônimos

Essa seção identifica as convenções de formato usadas para representar informações e terminologias tendo um significado específico. Os significados de abreviaturas e acrônimos usados neste documento são introduzidos também nesta seção.

1.1.2.1 Convenções

As convenções introduzidas nesta seção são usadas na descrição das operações especiais do CC sobre os requisitos de segurança e também para distinguir texto com significado específico. Estas convenções são consistentes com aquelas definidas no CC.

O CC permite que várias operações sejam executadas sobre requisitos funcionais: atribuição (*assignment*), iteração (*iteration*), refinamento (*refinement*) e seleção (*selection*) são definidas no parágrafo 2.1.4 da parte 2 do CC.

- A operação de *assignment* é usada para atribuir um valor específico a um parâmetro não especificado, tal como o tamanho de uma senha. Um *assignment* é indicado representando o valor entre colchetes [valor(es) atribuído(s)].

- A operação de *refinement* é usada para adicionar detalhes a um requisito, e dessa forma restringir um requisito. O refinamento dos requisitos de segurança é denotado em **negrito.**
- A operação de *selection* é usada para escolher um ou mais ítens de um lista ordenada para limitar o escopo de um elemento de componente. Seleções são denotadas em *texto itálico sublinhado*.
- Requisitos de garantia e funcionais aos quais é aplicada a operação de iteração (*iteration*), recebem identificadores únicos pela adição de um número de iteração dentro de parênteses ao identificador de requisito base, por exemplo, FMT_MOF.1.1 (1) e FMT_MOF.1.1 (2).

Texto em itálico é usado para títulos de documentos oficiais e texto para dar maior ênfase.

1.1.2.2 Terminologia

No Critério Comum, muitos termos são definidos na seção 2.3 da parte 1 (ISO/IEC 15408-1, 1999). Os seguintes termos são um subconjunto daquelas definições. Eles estão listados aqui para auxiliar o leitor do *Security Target*.

- ***Usuário*** – Qualquer entidade externa que interage com o TOE.
- ***Role*** – Um conjunto de regras pré-definidas que estabelecem as interações permitidas entre o usuário e o TOE.
- ***Política de Seguranca do TOE (TSP)*** – É o conjunto de regras usadas para mediar os acessos dos usuários aos objetos protegidos do TOE; define se a ação a ser executada pelo usuário é permitida ou proibida. Pode ser discricionária ou obrigatória.
- ***Política de Controle de Acesso Discricionária (DAC – Discretionary Access Control)*** – é caracterizada como a política que permite usuários e administradores autorizados controlar o acesso aos objetos com base nas suas identidades individuais ou *membership* de grupos.
- ***Política de Controle de Acesso Obrigatória (MAC – Mandatory Access Control)*** – É um conjunto de regras que determina o acesso baseado em uma sensibilidade (por exemplo, SECRETO) ou categoria (por exemplo, PESSOAL, MÉDICO) da informação que está sendo acessada e da autoridade de acesso do usuário que está requisitando a informação. A sensibilidade das informações e os direitos de acesso dos usuários são identificadas por marcas chamadas *rótulos de sensibilidade* (*sensitivity labels*). A combinação de uma classificação hierárquica e um conjunto de categorias não-

hierárquicas que representam a sensibilidade da informação é chamada *nível de segurança* (*security level*).

Além das definições genéricas acima listadas, o *Security Target* fornece as seguintes definições especializadas:

- ***Administrador Autorizado*** – Uma função a qual os usuários humanos podem estar associados para administrar os parâmetros de segurança do TOE. Tais usuários não estão sujeitos a quaisquer requisitos de controle de acesso e uma vez autenticados ao TOE, são confiáveis e não comprometem a política de segurança garantida pelo TOE.

1.1.2.3 Acrônimos

As seguintes abreviaturas do CC são usadas neste *Security Target*:

CC	Common Criteria for Information Technology Security Evaluation
EAL	Evaluation Assurance Level
IT	Tecnologia de Informação
SFP	Política de Função de Segurança
ST	Security Target
TOE	Target of Evaluation
TSC	Controle do Escopo da TSF
TSF	Funções de Segurança do TOE
TSP	Política de Segurança do TOE

1.1.3 Overview do TOE

Conforme visto em capítulos anteriores, o *Esquema de Autorização JaCoWeb DiscMand* é composto pelos seguintes módulos:

- Um *Serviço de política* (*PoliCap*), desenvolvido tendo como suporte a ferramenta JacORB v. 1.0 beta 13 (BROSE, 1997). Este serviço fornece interfaces para o desenvolvimento de aplicações que utilizam políticas de controle de acesso discricionárias e obrigatórias no contexto CORBAsec (conforme definido nos capítulos 4 e 5).
- Um *applet de aplicação* desenvolvido em Java JDK 1.2.1, assinado, que realiza o acesso a um servidor de aplicação CORBA.
- Um servidor de aplicação CORBA, desenvolvido tomando como base o JacORB v. 1.0 beta 13 (BROSE, 1997), que fornece serviços aos clientes representados pelos *applets* Java.
- Um pacote (conjunto de classes) *JaCoWebSecurity*, desenvolvido em Java e CORBA, que implementa, seguindo o nível de segurança *SecurityLevel2*, as interfaces e objetos padronizados do CORBAsec.

A política de segurança, presente no serviço de política *PoliCap*, é definida como global em um sistema distribuído em termos de atributos de privilégio dos principais e atributos de controle dos recursos do sistema, conforme descrito no capítulo 4. Uma vez definida através das interfaces administrativas do *PoliCap*, a política de segurança pode ser usada por aplicações no sentido de prover segurança nas interações entre *applets* e servidores de aplicação, em um ambiente de programação distribuída de larga escala.

1.1.4 Conformidade com o padrão ISO 15408

O TOE *JaCoWeb* está em conformidade com o *Protection Profile* designado *Labeled Security Protection Profile* (*LSPP*), Version 1.b, 8 October 1999 (NSA, 1999), registrado pela Agência de Segurança Nacional dos Estados Unidos (NSA – *National Security Agency*) – PPRegistry@gibraltar.ncsc.mil.

1.2 Descrição do TOE

Essa seção fornece o contexto para a avaliação do TOE identificando o tipo do produto e descrevendo a configuração avaliada.

1.2.1 Arquitetura

O Esquema de Autorização *JaCoWeb DiscMand* é composto pelos seguintes módulos:

- O serviço de política *PoliCap*;
- *Applets* de aplicação;
- Servidores de aplicação CORBA;
- Um pacote *JaCoWebSecurity*.

Os *applets* de aplicação interage com o servidor de nomes *CosNaming* CORBA para obter a IOR do servidor de aplicação e iniciar requisições de serviços.

O serviço de política *PoliCap* implementa as políticas de segurança do sistema, ambas discrionárias e obrigatórias, usando os objetos de serviço *DomainAccessPolicy* e *RequiredRights* do CORBAsec.

O administrador do sistema usa o serviço de política *PoliCap* para definir os objetos de política e de direitos requeridos, usando suas interfaces administrativas.

Servidores de aplicação CORBA fornecem serviços aos *applets* de aplicação.

O pacote *JaCoWebSecurity* supre as classes que implementam os objetos usados para implementar o nível de segurança *SecurityLevel2* definido pelo CORBAsec. Os controles

criptográficos são implementados por classes que usam a API do pacote IAIK-JCE (GRAZ, 1999b) e iSaSiLk (GRAZ, 1999a).

1.2.2 Escopo e Limites da Configuração Avaliada

Essa seção fornece uma descrição geral do escopo lógico e físico e dos limites do TOE. Considera-se implementadas as políticas discricionárias *do Esquema de Autorização JaCoWeb DiscMand* (http://www.lcmi.ufsc.br/jacoweb) (capítulo 4). Portanto, para fins de avaliação, será considerada parte do TOE implementada no ambiente real, como os *applets* de aplicação, servidores de aplicação e o pacote *JaCoWebSecurity*. O serviço de política *PoliCap*, tem sua implementação simplificada e portanto, será avaliado com relação ao esquema definido. As políticas obrigatórias também são avaliadas com base no esquema de autorização definido.

1.2.2.1 Escopo e Limites Físicos

O escopo físico do TOE inclui os elementos de *hardware* e *software* identificados na Tabela 1.1.

Componentes	Itens
Software	Pacote *JaCoWebSecurity* Versão 1.0
	Esquema de Autorização *JaCoWeb DiscMand* Versão 1.0 (*applets* de aplicação e servidores de aplicação que usam o pacote *JaCoWebSecurity*)
	JacORB v. 1.0 beta 13
	IAIK-JCE 2.5
	Netscape Communicator 4.5
	Java JDK 1.2.1 da Sun
	Microsoft Windows 95 e FreeBSD 2.2.8
	Apache Web Server 1.3.9
Hardware	2 computadores Intel P133 64Mb em rede
	30 Mb espaço em disco rígido (pelo menos)

Tabela 1.1 – Componentes de *Software/Hardware*.

1.2.2.2 Escopo e Limites Lógicos

O TOE fornece as seguintes características de segurança:

- **Auditoria**: A geração de dados de auditoria são feitos via arquivos de log's e implementados conforme política de auditoria introduzida no modelo CORBAsec. Esta política é definida pelo administrador do sistema.
- **Suporte Criptográfico**: Os controles criptográficos são implementados pelo uso do *iSaSiLk* (GRAZ, 1999a), versão do SSL em Java, durante o estabelecimento da conexão segura usando o CORBA. O pacote IAIK-JCE (GRAZ, 1999b) implementa os algoritmos criptográficos usados pelo *iSaSiLk*. O algoritmo de chave pública, usado para autenticação mútua e

distribuição de chaves, é o RSA de 1024 bits. Para a cifragem dos dados é usado o algoritmo simétrico 3DES_EDE (cifragem em blocos) e a função *hash* de segurança SHA para a computação do MAC (capítulo 4). A assinatura dos *applets*[44] clientes é feita usando recursos da linguagem Java JDK 1.2.1.

- **Proteção de Dados do Usuário**: As políticas de proteção de dados do usuário são implementadas a partir dos objetos de política de segurança do sistema, centralizados no serviço de política *PoliCap*. Além desses objetos, os objetos locais de política e os objetos de decisão de acesso e de sessão (*PolicyCurrent*, *Current* e *SecurityManager*), implementam as funções de controle de acesso do sistema, localmente em cada máquina que participa das comunicações. A autenticação mútua entre as partes comunicantes é estabelecida através do estabelecimento da associação segura via CORBA, usando o SSL, entre objetos de aplicação cliente e servidor. A cada nova associação segura estabelecida entre dois objetos quaisquer, novos objetos de serviço de segurança são instanciados por sessão, impedindo que qualquer informação residual (prévia/permanente) seja transferida em novas sessões.
- **Identificação e Autenticação**: Há três fases de autenticação presentes no TOE: a autenticação *do código móvel ou applet Java*, a autenticação do *principal* e a autenticação *mútua entre cliente e servidor* (autenticação da sessão do modelo de segurança CORBAsec). A autenticação do *applet* Java, que é carregado pelo *browser* e representa o cliente no ambiente considerado, usa os recursos de assinatura de applets provido pelo ambiente Java JDK 1.2. A *autenticação do principal* pelo modelo CORBAsec é realizada pelo pacote *JaCoWebSecurity*, usando certificados X.509v3 dos usuários. Estes certificados também são usados para a autenticação mútua entre cliente e servidor (*autenticação da sessão* do modelo CORBAsec), que é feita pelo próprio SSL, durante o estabelecimento da associação segura do CORBAsec (capítulo 4).
- **Gerência da Segurança**: A gerência dos objetos de política e da relação entre usuários e atributos de privilégio, é realizada por administradores autorizados do sistema. Objetos da aplicação também atuam para definir os objetos de política na parte que lhes cabe.
- **Proteção das Funções de Segurança**: As funções de segurança não podem ser evitadas ou modificadas pois as políticas de segurança são aplicadas de forma transparente a uma aplicação, pelo próprio ORB, através dos objetos de serviço do CORBAsec, presentes no pacote *JaCoWebSecurity*.

[44] Um relatório do processo de assinatura do *applet* cliente está disponível no endereço http://www.lcmi.ufsc.br/jacoweb/documentos.

1.2.3 Contexto de Aplicação

O TOE avaliado é usado para prover programação distribuída segura em ambientes de pequena e larga escala. Possui interface gráfica representada pelo *applet* de aplicação instalada na máquina do cliente. Esse *applet* é carregado a partir de um servidor Web. O serviço de política e servidores de aplicação são instaciados no momento em que um cliente inicia requisições de serviços. O pacote *JaCoWebSecurity* está presente na máquina do cliente e na máquina dos servidores de aplicação para fornecer acesso aos objetos de serviço do CORBAsec.

1.3 Ambiente de Segurança do TOE

O TOE deve ser usado em ambientes nos quais as informações sensíveis a serem processadas possuam ou não níveis de sensibilidade associados, estabelecidos pela política de segurança do ambiente.

A identificação de ameaças conhecidas ou aceitas sobre os recursos do sistema determinam as formas de proteção que o *Esquema de Autorização JaCoWeb DiscMand* e seus ambientes devem fornecer. Este contexto todo forma o ambiente de segurança do TOE. Para apresentar o ambiente de segurança do *Esquema de Autorização JaCoWeb DiscMand*, suposições sobre esse ambiente são fornecidos na seqüência.

1.3.1 Suposições

Essa seção descreve os aspectos de segurança do ambiente no qual o TOE será usado. Isso inclui informações sobre aspectos físicos, pessoais e de conectividade do ambiente.

Um TOE que declara conformidade com o *LSPP* (*Labeled Security Protection Profile* (NSA, 1999)), deve fornecer medidas de segurança em um ambiente cooperativo apenas se instalado, gerenciado e usado corretamente. O ambiente operacional deve ser gerenciado de acordo com a documentação dos requisitos de garantia para liberação, operação e guias de usuário/administrador.

O TOE declara as seguintes suposições na seção 3.3 do LSPP.

1.3.1.1 Suposições Físicas

Assume-se que existem as seguintes condições físicas descritas na Tabela 1.2.

Nome	Descrição
A.LOCATE	Os recursos de processamento do TOE estarão localizados dentro de facilidades de controle que previnem o acesso físico não autorizado.
A.PROTECT	Os *hardwares* e *softwares* críticos para a garantia da política de segurança do TOE serão protegidos de modificação física não autorizada.

Tabela 1.2 – Suposições Físicas conforme LSPP.

1.3.1.2 Suposições sobre Pessoal

Assume-se que existem as seguintes condições de pessoal descritas na Tabela 1.3.

Nome	Descrição
A.MANAGE	Existem um ou mais indivíduos competentes para gerenciar o TOE e as informações de segurança que ele contém.
A.NO_EVIL_ADM	As pessoas relacionadas com a administração do sistema são cuidadosas, responsáveis e nada hostis e seguem as instruções fornecidas pela documentação de administração.
A.COOP	Usuários autorizados possuem a necessária autorização para acessar informações gerenciadas pelo TOE e espera-se que atuem de forma cooperativa em um ambiente benigno.

Tabela 1.3 – Suposições de Pessoal conforme LSPP.

1.3.1.3 Suposições Procedurais

A habilidade do TOE em garantir o objetivo da política de segurança organizacional, especialmente considerando controles de acesso obrigatórios, é dependente do estabelecimento de alguns procedimentos. Existem os seguintes procedimentos de controle como descritos na Tabela 1.4.

Nome	Descrição
A.CLEARANCE	Existem procedimentos para fornecer autorização aos usuários para o acesso em níveis de segurança específicos.
A.SENSITIVITY	Existem procedimentos para estabelecer o nível de segurança de todas as informações importadas para o sistema, para estabelecer o nível de segurança de todos os dispositivos periféricos (por exemplo, impressoras, *drives* de disco) ligadas ao TOE, e também, para rotular todas as saídas geradas com um rótulo de sensibilidade.

Tabela 1.4 – Suposições Procedurais conforme LSPP.

1.3.1.4 Suposições de Conectividade

Existem as seguintes condições de conectividade que são apresentadas na Tabela 1.5.

Nome	Descrição
A.PEER	Assume-se que outros sistemas com os quais o TOE se comunica estão sob o mesmo controle de gerenciamento e operam sob as mesmas restrições de política de segurança, dentro de um único domínio de gerenciamento.
A.CONNECT	Todas as conexões a dispositivos periféricos residem dentro de facilidades de acesso controladas. Trata-se neste contexto a segurança relacionada apenas com a manipulação do TOE através dos seus pontos de acesso autorizados.

Tabela 1.5 – Suposições de Conectividade conforme LSPP.

1.3.2 Ameaças

O TOE foi concebido para tratar as ameças descritas na Tabela 1.6.

Nome	Descrição
T.ACCESS	Um usuário autorizado do TOE pode acessar informações ou recursos sem ter permissão do proprietário ou responsável pela informação ou recurso.
T.CAPTURE	Um invasor pode observar ou capturar dados sendo transmitidos pela rede.
T.INTEGRITY	A integridade da informação pode ser comprometida devido a erros do usuário, de transmissão ou falhas de *hardware*.
T.SECRET	Um usuário autorizado do TOE pode, intencional ou acidentalmente, observar informações armazenadas no TOE para as quais não possui autorização.
T.IMPERSON	Um invasor (interno ou externo) pode obter acesso autorizado às informações ou recursos se fazendo passar por um usuário autorizado do TOE.

Tabela 1.6 – Ameaças tratadas pelo TOE.

O TOE também pode estar sujeito a ameaças operacionais como a descrita na Tabela 1.7.

Nome	Descrição
T.TUSAGE	O TOE pode ser inadvertidamente configurado, usado e administrado de forma insegura por pessoas autorizadas ou não autorizadas.

Tabela 1.7 – Ameaças tratadas pelo Ambiente Operacional.

1.3.3 Políticas de Segurança Organizacionais

Uma política de segurança organizacional é um conjunto de regras ou procedimentos impostos por uma organização sobre operações para proteger seus dados sensíveis. As políticas organizacionais descritas nesse item estão em conformidade com LSPP e são descritas na Tabela 1.8.

Nome	Descrição
P.AUTHORIZED_ USERS	Apenas usuários autorizados às informações do sistema tem acesso ao sistema.
P.NEED_TO_KNOW	O sistema deve limitar o acesso, a modificação e a destruição da informação, em recursos protegidos, dos usuários autorizados.
P.ACCOUNTABILITY	Os usuários do sistema devem ser responsabilizados por suas ações dentro do sistema.
P.CLASSIFICATION	O sistema deve regrar as formas de acesso tomando como base os rótulos de sensibilidade das informações contidas em objetos e a habilitação dos usuários, representada pelos sujeitos que requisitam os acessos correspondentes. As regras de acesso aplicadas impedem o sujeito de ter acesso a informações com sensibilidade maior do que a sua habilitação e impedem também o sujeito de causar uma modificação de informações em níveis de sensibilidade inferiores. A determinação da classificação da informação e da habilitação dos usuários está fora do escopo do sistema de tecnologia de informação. Essa classificação é essencialmente baseada em regras da organização e nas funções exercidas pelos usuários dentro da organização.

Tabela 1.8 – Política de Segurança Organizacional.

Além das regras acima citadas, define-se mais uma regra na Tabela 1.9.

Nome	Descrição
P.DAC	O direito para acessar objetos específicos é determinado tomando como base a identidade do sujeito que deseja o acesso e os direitos de acesso implícitos e explícitos do objeto, fornecidos ao sujeito pelo proprietário do objeto.

Tabela 1.9 – Regra adicional da Política de Segurança Organizacional.

1.4 Objetivos de Segurança

Esta seção define os objetivos de segurança das TSFs (funções de segurança do TOE) e do seu ambiente de suporte. O propósito dos objetivos de segurança é detalhar uma resposta planejada aos problemas ou ameaças à segurança do sistema. Ameaças podem ser direcionadas contra o TOE ou o ambiente de segurança ou ainda contra ambos. Portanto, o CC identifica duas categorias de objetivos de segurança: objetivos de segurança do TOE e objetivos de segurança do ambiente.

1.4.1 Objetivos de Segurança do TOE

O TOE possui os objetivos de segurança da tecnologia da informação (IT) delineados na seção 4 do LSPP e descritos na Tabela 1.10.

Nome	Descrição
O.AUTHORIZATION	As TSFs devem garantir o acesso ao TOE e aos seus recursos apenas a usuários autorizados.
O.DISCRETIONARY_ ACCESS	As TSFs devem controlar o acesso aos recursos baseadas na identidade dos usuários. As TSFs devem permitir aos usuários autorizados a possibilidade de especificar quais recursos podem ser acessados por quais usuários.
O.MANDATORY_ ACCESS	As TSFs devem controlar o acesso aos recursos baseadas na sensibilidade e categorias da informação que está sendo acessada e na habilitação do sujeito que requisita o acesso à informação.
O.AUDITING	As TSFs devem registrar as ações de segurança relevantes dos usuários do TOE. As TSFs devem apresentar essas informações para administradores autorizados.
O.RESIDUAL_ INFORMATION	As TSFs devem garantir que informações contidas em um recurso protegido não sejam liberadas quando o recurso é reutilizado (*recycled*).
O.MANAGE	As TSFs devem fornecer todas as funções e facilidades necessárias para suportar administradores autorizados que sejam responsáveis pelo gerenciamento da segurança do TOE.
O.ENFORCEMENT	As TSFs devem ser projetadas e implementadas de forma a garantir que as políticas de segurança organizacionais sejam aplicadas no ambiente destino.

Tabela 1.10 – Objetivos de Segurança do TOE.

1.4.2 Objetivos de Segurança do Ambiente

Um TOE em conformidade com o LSPP é dito completo e autosuficiente, sendo, portanto, não dependente de outros produtos para operar corretamente. Entretanto, certos objetivos relacionados com o ambiente operacional (no caso assumido como genérico) devem ser respeitados e são descritos na Tabela 1.11.

Nome	Descrição
O.INSTALL	Os responsáveis pelo TOE devem garantir que o TOE é entregue, instalado, gerenciado e operado de maneira que se mantenham os objetivos de segurança da IT.
O.PHYSICAL	Os responsáveis pelo TOE devem garantir que as partes críticas do TOE para a política de segurança são protegidas de ataques físicos que possam comprometer os objetivos de segurança da IT.

Tabela 1.11 – Objetivos de Segurança do Ambiente Operacional.

1.5 Requisitos de Segurança da Tecnologia de Informação

Os requisitos de segurança da tecnologia de informação (IT) incluem:

- Requisitos de segurança do TOE e
- Requisitos de segurança para o ambiente de tecnologia de informação (opcional).

- Os requisitos de segurança para o ambiente são para o *hardware*, o *software* ou o *firmware* externos ao TOE sobre as quais reside a satisfação dos objetivos de segurança do TOE (item 1.5.3).

O CC divide os requisitos de segurança do TOE em duas categorias:

- Requisitos funcionais de segurança (*SFRs – Security Functional Requirements*), isto é, requisitos para as funções de segurança tais como controle de acesso às informações, auditoria, identificação e autenticação (item 1.5.1).
- Requisitos de garantia de segurança (*SARs – Security Assurance Requirements*), que fornecem a base para a confiança de que o TOE atinge os seus objetivos de segurança (por exemplo, documentos guias, testes, declaração de vulnerabilidades) (item 1.5.2).

1.5.1 Requisitos Funcionais de Segurança do TOE

Essa seção apresenta os SFRs para o TOE. São descritos apenas nominalmente os requisitos funcionais que encontram-se em LSPP (NSA, 1999), visto que esse TOE segue conformidade com o LSPP. Se o ST declara conformidade com os requisitos de um PP sem necessidade de esclarecimentos, então a referência ao PP é suficiente para definir e justificar os objetivos e requisitos do TOE. A declaração dos conteúdos do PP é desnecessária[45]. Mesmo assim, alguns comentários sobre cada uma das classes de requisitos funcionais são feitos aqui.

As tabelas Tabela 1.12, Tabela 1.13, Tabela 1.14, Tabela 1.15 e Tabela 1.16 apresentam os requisitos funcionais presentes no LSPP, aos quais este TOE declara conformidade. São descritos em maiores detalhes as adições feitas ao LSPP, relativas aos controles criptográficos (Tabela 1.17).

Nome da Classe	**Identificador do Componente Funcional**	**Nome do Componente Funcional**
Classe FAU – Auditoria de Segurança	FAU_GEN.1	Geração de Dados de Auditoria
	FAU_GEN.2	Associação da Identidade do Usuário
	FAU_SAR.1	Revisão da Auditoria
	FAU_SAR.2	Revisão da Auditoria Restrita
	FAU_SAR.3	Revisão da Auditoria Selecionada
	FAU_SEL.1	Auditoria Seletiva
	FAU_STG.1	Garantias de Disponibilidade de Dados de Auditoria
	FAU_STG.3	Ação no caso da possível perda dos Dados de Auditoria
	FAU_STG.4	Prevenção da Perda dos Dados de Auditoria

Tabela 1.12 – Classe FAU.

[45] CC Parte 1, Anexo C, par. C.2.8, b.

A classe FAU estabelece procedimentos de auditoria no TOE e envolve o reconhecimento, registro, armazenamento e análise de informações relacionadas às atividades de segurança relevantes (isto é, atividades controladas pela TSP). Os registros resultantes de auditoria podem ser examinados para determinar quais atividades de segurança relevantes aconteceram e quem (qual usuário) é responsável pelas mesmas.

Nome da Classe	**Identificador do Componente Funcional**	**Nome do Componente Funcional**
Classe FDP – Proteção dos Dados dos Usuários	FDP_ACC.1	Política de Controle de Acesso Discricionária
	FDP_ACF.1	Funções de Controle de Acesso Discricionárias
	FDP_ETC.1	Exportação de Dados de Usuário Não Rotulados
	FDP_ETC.2	Exportação de Dados de Usuário Rotulados
	FDP_IFC.1	Política de Controle de Acesso Obrigatória
	FDP_IFF.2	Funções de Controle de Acesso Obrigatórios
	FDP_ITC.1	Importação de Dados de Usuário Não-Rotulados
	FDP_ITC.2	Importação de Dados de Usuário Rotulados
	FDP_RIP.2	Proteção às Informações Residuais dos Objetos

Tabela 1.13 – Classe FDP.

A classe FDP contém famílias especificando os requisitos das funções de segurança do TOE e das políticas de funções de segurança relacionadas ao processamento de dados de usuário. Essa classe é dividida em quatro grupos de famílias que tratam dos dados de usuário dentro do TOE, durante a importação, exportação e armazenamento dos mesmos bem como dos atributos de segurança diretamente relacionados aos dados do usuário.

Nome da Classe	**Identificador do Componente Funcional**	**Nome do Componente Funcional**
Classe FIA – Identificação e Autenticação	FIA_ATD.1	Definição dos Atributos dos Usuários
	FIA_SOS.1	Poder (*Strength*) dos Dados de Autenticação
	FIA_UAU.1	Autenticação
	FIA_UAU.7	*Feedback* da Autenticação Protegida
	FIA_UID.1	Identificação
	FIA_USB.1	Ligação (*binding*) entre o usuário e sujeito

Tabela 1.14 – Classe FIA.

A classe FIA trata dos requisitos das funções que estabelecem e verificam a identidade do usuário. Identificação e autenticação são necessárias para garantir que os usuários estão associados com atributos de segurança devidos (por exemplo, identidade, grupos, *roles*, níveis de segurança ou de integridade). As famílias dessa classe tratam da verificação da identidade dos usuários, determinação da sua autoridade para interagir com o TOE, e da correta associação dos atributos de segurança para cada usuário autorizado. Outras classes de requisitos (por exemplo, FDP e FAU) são dependentes da correta identificação e autenticação dos usuários a fim de serem efetivas.

Nome da Classe	Identificador do Componente Funcional	Nome do Componente Funcional
Classe FMT – Gerência da Segurança	FMT_MSA.1	Gerência dos Atributos de Segurança do Objeto
	FMT_MSA.3	Inicialização dos Atributos Estáticos
	FMT_MTD.1 (1)	Gerência da trilha (*trail*) de Auditoria
	FMT_MTD.1 (2)	Gerência dos Eventos de Auditoria
	FMT_MTD.1 (3)	Gerência dos Atributos dos Usuários
	FMT_MTD.1 (4)	Gerência dos Dados de Autenticação
	FMT_REV.1 (1)	Revogação de Atributos de Usuário
	FMT_REV.1 (2)	Revogação de Atributos de Objeto
	FMT_SMR.1	Papéis (*roles*) na gerência da segurança

Tabela 1.15 – Classe FMT.

A classe FMT especifica a gerência de vários aspectos das TSF: atributos de segurança (listas de controle de acesso, por exemplo), dados e funções das TSF, definição dos *roles* de segurança dos usuários.

Nome da Classe	Identificador do Componente Funcional	Nome do Componente Funcional
Classe FPT – Proteção das Funções de Segurança do TOE	FPT_AMT.1	Teste da Máquina Abstrata
	FPT_RVM.1	Mediação de Referência
	FPT_SEP.1	Separação de Domínios
	FPT_STM.1	*Time Stamps* Confiáveis

Tabela 1.16 – Classe FPT.

A classe FPT contém famílias de requisitos funcionais que se relacionam com a proteção dos dados das TSFs. Componentes desta classe são necessários para fornecer requisitos que garantam a não modificação e evitem o *bypass* (evitar a não execução) das funções que implementam a política de segurança do TOE. Do ponto de vista dessa classe, existem três partes significativas das TSFs:

- a *máquina abstrata* das TSF's, que é a máquina virtual ou física sobre a qual a implementação das TSFs específicas, sob avaliação, executam;
- a *implementação* das TSF's que executam na máquina abstrata e implementam os mecanismos que garantem a TSP e;
- os *dados* das TSF's, que são bancos de dados administrativos que guiam a aplicação das TSP.

Nome da Classe	Identificador do Componente Funcional	Nome do Componente Funcional
Classe FCS – Suporte Criptográfico	FCS_CKM.3	Acesso a chaves criptográficas
	FCS_CKM.4	Destruição de chaves criptográficas
	FCS_COP.1	Operação criptográfica

Tabela 1.17 – Classe FCS.

A classe FCS é usada quando o TOE emprega funções criptográficas. Essa classe é composta por duas famílias: FCS_CKM para gerência de chaves criptográficas e FCS_COP para operação criptográfica.

Segue o detalhamento da funções criptográficas, conforme parte 2 do CC (ISO/IEC 15408-2, 1999).

FCS_CKM.3 Acesso a chaves criptográficas

FAU_CKM.3.1 As TSF devem executar [um tipo de acesso à chave criptográfica] de acordo com o método de acesso a chave criptográfica especificado [método de acesso a chave criptográfica] que segue a seguinte [lista de padrões].

FCS_CKM.4 Destruição de chaves criptográficas

FAU_CKM.4.1 As TSF devem destruir as chaves criptográficas de acordo com o método de destruição de chaves criptográficas especificado [método de destruição de chaves criptográficas] que segue a seguinte [lista de padrões].

FCS_COP.1 Operação Criptográfica

FAU_CKM.1.1 As TSF devem executar [lista de operações criptográficas] de acordo com um algoritmo criptográfico especificado [algoritmo criptográfico] e tamanhos de chaves criptográficas [tamanhos de chaves criptográficas] que segue a seguinte [lista de padrões].

1.5.2 Requisitos de Garantia (assurance) de Segurança do TOE

A Tabela 1.18 identifica os componentes de garantia de segurança identificados a partir da parte 3 do CC (ISO/IEC 15408-3, 1999): *Security Assurance Requirements*, EAL 3, em conformidade com LSPP.

A classe ACM ajuda a garantir que a integridade do TOE seja preservada, pelo uso de disciplina e controle dos processos de refinamento e modificação do TOE e de outras informações relacionadas. A gerência da configuração impede modificações, adições ou remoções não autorizadas do TOE, fornecendo a garantia de que o TOE e a documentação utilizada para avaliação são as mesmas usadas para a distribuição do TOE.

Nome da Classe de Garantia	Identificador do Componente de Garantia	Nome do Componente de Garantia
Classe ACM – Gerência de Configuração	ACM_CAP.3	Controles de Autorização
	ACM_SCP.1	Cobertura da Gerência de Configuração do TOE
Classe ADO – Liberação (*delivery*) e Operação	ADO_DEL.1	Procedimentos de liberação
	ADO_IGS.1	Instalação, geração e procedimentos iniciais
Classe ADV – Desenvolvimento	ADV_FSP.1	Especificação funcional informal
	ADV_HLD.2	Aplicação da segurança no projeto de alto-nível
	ADV_RCR.1	Demonstração da correspondência informal
	ADV_SMP.1	Modelagem da política de segurança
Classe AGD – Documentos Guias	AGD_ADM.1	Guia do administrador
	AGD_USR.1	Guia do usuário
Classe ALC – Suporte ao Ciclo de Vida	ALC_DVS.1	Identificação das medidas de segurança
Classe ATE – Testes	ATE_COV.2	Análise de cobertura
	ATE_DPT.1	Testes: projeto de alto nível
	ATE_FUN.1	Testes funcionais
	ATE_IND.2	Exemplo de teste independente
Classe AVA – Declaração de Vulnerabilidade	AVA_MSU.1	Exame dos guias
	AVA_SOF.1	Poder (*strength*) da avaliação da função de segurança do TOE
	AVA_VLA.1	Análise de vulnerabilidades feita pelo desenvolvedor

Tabela 1.18 – Requisitos de Garantia do TOE conforme LSPP.

A classe ADO define requisitos para garantia através da adoção de um modelo de ciclo de vida bem definido para todas as fases de desenvolvimento do TOE, incluindo procedimentos e políticas de correção de falhas, correto uso de técnicas e ferramentas e as medidas de segurança usadas para proteger o ambiente de desenvolvimento. Essa classe define requisitos para as medidas, procedimentos e padrões referentes a liberação, instalação e uso operacional seguros do TOE, garantindo que a proteção de segurança oferecida pelo TOE não é comprometida durante a transferência, instalação, inicialização e operação.

A classe ADV define requisitos para o refinamento das funções de segurança TSF's obtidas da especificação sumária do TOE no ST até a implementação real. Cada uma das representações resultantes das TSF's fornecem informações para auxiliar o avaliador a determinar se os requisitos funcionais do TOE foram atendidos.

A classe AGD define requisitos direcionados para o entendimento, cobertura e completitude da documentação operacional fornecida pelo desenvolvedor. Essa documentação, que fornece duas categorias de informações, para usuários e para administradores, é um fator importante na operação segura do TOE.

A classe ALC define requisitos para a garantia através da adoção de um modelo de ciclo de vida bem definido para todos os passos do desenvolvimento do TOE, incluindo procedimentos e

políticas de remediação de falhas, uso correto de ferramentas e técnicas e as medidas de segurança usadas para proteger o ambiente de desenvolvimento.

A classe ATE define requisitos de testes que demonstram que as TSF's satisfazem os requisitos funcionais de segurança do TOE. Define testes de cobertura, de profundidade, testes funcionais e independentes.

A classe AVA define requisitos direcionados para a identificação e exploração das vulnerabilidades. Especificamente, ela trata das vulnerabilidades introduzidas na construção, operação, uso mal intencionado ou configuração incorreta do TOE.

1.5.3 Requisitos de Segurança para o ambiente de tecnologia de informação

O TOE (*JaCoWeb*) não possui requisitos de segurança definidos para o ambiente de tecnologia de informação.

1.6 Especificação Sumária do TOE

Essa seção apresenta um resumo funcional do TOE, descrevendo suas funções de segurança implementadas e as medidas de garantia aplicadas para garantir suas corretas implementações.

1.6.1 Funções de Segurança do TOE

As funções de segurança executadas pelo TOE devem ter suas descrições para auxiliar na sua avaliação. Neste sentido é fornecida a declaração (*traceability*) dos requisitos de funções de segurança (SFR's).

1.6.1.1 Gerência da Segurança (FMT)

O serviço de política *PoliCap* mantém os atributos de segurança do ambiente para todos os administradores, usuários e objetos do sistema. As interfaces *DomainAccessPolicy* e *RequiredRights* oferecem operações que são usadas por aplicações administrativas para a definição da política de segurança discricionária e/ou obrigatória do domínio (objeto *DomainAccessPolicy*) e do objeto de direitos requeridos (objeto *RequiredRights*).

O objeto de política de auditoria *Audit Policy* determina os dados a serem examinados durante uma sessão. Esse objeto é definido pelo administrador autorizado do sistema.

As seguintes funções administrativas podem ser executadas por usuários autenticados e autorizados:

- Criar, remover, modificar e verificar os atributos dos usuários e dos objetos;

- Modificar e definir o número de tentativas de autenticação permitidas por administradores a usuários normais do TOE;
- Restabelecer a capacidade de autenticação para usuários que ultrapassaram as tentativas de autenticação sem sucesso;
- Arquivar, criar, remover, revisar e esvaziar a trilha (*trail*) de auditoria.

Os **requisitos funcionais satisfeitos no *JaCoWeb*** em relação à gerência da segurança (ver Tabela 1.15) são: FMT_MSA.1, FMT_MSA.3, FMT_MTD.1 (1), FMT_MTD.1 (2), FMT_MTD.1 (3), FMT_MTD.1 (4), FMT_REV.1 (1), FMT_REV.1 (2) e FMT_SMR.1.

1.6.1.2 Identificação e Autenticação (FIA)

A identificação e autenticação tratam da definição de atributos do usário, da 'força' dos dados de autenticação, da autenticação, da proteção dos dados usados no momento da autenticação, da identificação e da ligação entre usuário e sujeito.

As três fases de autenticação presentes no TOE *JaCoWeb* - a autenticação *do código móvel ou applet Java*, a autenticação do *principal* e a autenticação *mútua entre cliente e servidor* - já foram discutidas no capítulo 4 e brevemente na seção 6.3.1. A autenticação do *applet* Java, usa os recursos de assinatura de *applets* provido pelo ambiente Java JDK 1.2. A *autenticação do principal* pelo modelo CORBAsec é realizada pelo pacote *JaCoWebSecurity*, usando certificados X.509v3 dos usuários. Estes certificados também são usados para a autenticação mútua entre cliente e servidor (*autenticação da sessão* do modelo CORBAsec), que é feita durante o estabelecimento da associação segura do CORBAsec.

A proteção dos dados durante o processo de autenticação do principal a fim de gerar as credenciais é fornecida pela classe *Security* que impede que a aplicação acesse dados durante a fase de autenticação.

A 'força' dos dados de autenticação é medida verificando-se que existe baixa probabilidade de que os dados usados na autenticação possam ser forjados ou adivinhados. Nesse TOE, os dados de autenticação são fornecidos através de certificados X.509v3 fornecidos por uma terceira entidade (ITU-TX509, 1993).

Os **requisitos funcionais satisfeitos no *JaCoWeb*** em relação à identificação e autenticação (ver Tabela 1.14) são: FIA_ATD.1, FIA_SOS.1, FIA_UAU.1, FIA_UAU.7, FIA_UID.1 e FIA_USB.1.

1.6.1.3 Proteção de Dados de Usuário (FDP)

Os requisitos para a proteção dos dados do usuário são compostos pela política de controle de acesso discricionária, pela política de controle de acesso obrigatória, pela importação de dados de usuário rotulados e não-rotulados, pela exportação de dados de usuário rotulados e não-rotulados e pela proteção das informações residuais dos objetos.

As políticas de controle de acesso discricionária e obrigatória são implementadas pelo serviço de política *PoliCap*, descritas no capítulo 4 e no capítulo 5, que usa os objetos de serviço do CORBAsec. As funções de controle de acesso discricionárias e obrigatórias, que são responsáveis pela autorização da invocação, são implementadas pelos objetos *AccessDecision* locais em cada máquina participante da aplicação distribuída.

A importação e a exportação de dados relativos ao usuário, rotulados e não-rotulados, é controlada pelos objetos de política do CORBAsec no sistema.

A cada nova associação segura estabelecida entre dois objetos quaisquer, novos objetos de sessão são instanciados, impedindo que qualquer informação prévia (residente) seja transferida para futuras sessões.

Os **requisitos funcionais satisfeitos no *JaCoWeb*** em relação à proteção de dados de usuário (ver Tabela 1.13) são: FDP_ACC.1, FDP_ACF.1, FDP_ETC.1, FDP_ETC.2, FDP_IFC.1, FDP_IFF.2, FDP_ITC.1, FDP_ITC.2 e FDP_RIP.2.

1.6.1.4 Proteção das Funções de Segurança (FPT)

A proteção das funções de segurança TSFs (*TOE Security Functions*) é necessária para fornecer requisitos que garantam a não modificação e evitem o *bypass* (a não execução), das funções que implementam as políticas de segurança do TOE.

Do ponto de vista desta classe, existem três partes significativas que compõem as TSFs: a máquina abstrata das TSFs, a implementação e os dados das TSFs. A máquina abstrata das TSFs é definida pela máquina física ou virtual sobre a qual a implementação de uma TSF específica sob avaliação é executada. A implementação das TSFs concretiza os mecanismos que garantem as políticas de segurança do TOE e é executada sobre a máquina abstrata. Os dados das TSFs são bancos de dados administrativos que guiam a aplicação da política.

Os requisitos para a proteção das funções de segurança TSFs, de acordo com esta classe, são compostos pelo "teste da máquina abstrata", pela mediação de referência, pela separação de domínios e por *time-stamps* confiáveis. Todos esses requisitos servem para garantir a integridade e gerência dos mecanismos que fornecem as TSFs e garantir a integridade dos dados das TSFs.

O teste da máquina abstrata diz respeito a garantir que durante a inicialização do sistema, ou na sua operação normal ou ainda durante os pedidos de usuários autorizados, a máquina abstrata sobre a qual o TOE é executado opere de forma correta, sem quaisquer falhas de projeto. Os cuidados no desenvolvimento e operação do TOE *JaCoWeb*, com o uso de ferramentas de especificação e de metodologias de construção de *software*, nos permite assumir esta premissa de que nada de anormal ocorra para cobrir esse requisito funcional (seção 6.3.1).

O requisito funcional da mediação de referência trata o aspecto tradicional do monitor de referência, garantindo que todas as ações a serem verificadas de acordo com a política de segurança, sejam validadas pelas TSFs que implementam as políticas de segurança do ambiente.

No *JaCoWeb*, as TSFs, que são implementadas pelos objetos *AccessDecision* locais do CORBAsec em cada *site* do ambiente, garantem que as políticas de segurança não podem ser evitadas durante uma requisição da aplicação. A política de segurança aplicada a nível de ORB, como é o caso desse TOE, é executada automaticamente de forma transparente para a aplicação, através dos objetos de serviço do CORBAsec, presentes no pacote *JaCoWebSecurity*, pelo próprio ORB.

A separação de domínios diz respeito ao isolamento das TSFs, em relação às aplicações ou outros sujeitos não confiáveis, para prevenir a modificação de códigos ou estruturas de dados e a interferência durante a execução das TSFs. No *JaCoWeb*, este requisito funcional é implementado pelo uso dos interceptadores do modelo CORBAsec, impossibilitando a invocação das TSFs à sujeitos não confiáveis.

O requisito funcional dos *time-stamps*, que são usados pelas próprias TSFs, é necessário caso exista eventos de auditoria que são registrados pelo sistema. Neste caso, deve existir, no mínimo, notificações do relógio de forma a atribuir *time-stamps* a estes eventos. No *JaCoWeb*, os *time-stamps* são implementados pelo uso do relógio do sistema a fim de registrar os eventos de auditoria.

Os **requisitos funcionais satisfeitos no *JaCoWeb*** em relação à proteção das funções de segurança (ver Tabela 1.16) são: FPT_AMT.1, FPT_RVM.1, FPT_SEP.1 e FPT_STM.1.

1.6.1.5 Funções Criptográficas (FCS)

Os requisitos que são usados para definir as funções criptográficas são o gerenciamento de chaves criptográficas e a operação criptográfica propriamente dita.

Estas operações necessárias durante o estabelecimento e a duração de uma conexão segura no CORBAsec são providas pelo *iSaSiLk* (GRAZ, 1999a), versão do SSL em Java. Os algoritmos disponíveis via *iSaSiLk* são implementados no pacote IAIK-JCE (GRAZ, 1999b). O algoritmo de

chave pública, usado para autenticação mútua e distribuição de chaves, é o RSA de 1024 bits. Na cifragem dos dados utiliza-se o algoritmo simétrico de cifragem em bloco 3DES_EDE e a função *hash* de segurança SHA para a computação do MAC (capítulo 4). A assinatura dos *applets* clientes é feita usando recursos da linguagem Java JDK 1.2.1.

Os **requisitos funcionais satisfeitos no *JaCoWeb*** em relação à proteção das funções de segurança (ver Tabela 1.17) são: FCS_CKM.3, FCS_CKM.4 e FCS_COP.1.

1.6.1.6 Auditoria (FAU)

Os requisitos de auditoria contemplam a geração dos dados de auditoria, a associação da identidade do usuário, a revisão da auditoria, a auditoria seletiva, as garantias de disponibilidade de dados de auditoria, a ação no caso de perdas dos dados de auditoria e a prevenção da perda dos dados de auditoria.

A geração de dados de auditoria são feitos via arquivos de log's e implementados pela política de auditoria (objeto *AuditPolicy*) definida no modelo CORBAsec. Essa política é definida pelo administrador do sistema e é armazenada no serviço de políticas *PoliCap,* sendo carregada em tempo de ligação entre objetos cliente e servidor em ambos os lados.

Aplicações administrativas fornecem acesso aos dados de auditoria apenas para o administrador do sistema que pode revisar e selecionar dados de interesse. Depois de um certo tempo, o administrador é responsável por eliminar dados desnecessários dos arquivos de *log* gerados.

A auditoria no *JaCoWeb* usa os objetos de serviço do CORBAsec *AuditDecision*, *AuditPolicy* e *AuditChannel* (BLAKLEY, 1999, OMG, 2000a) e foi detalhada no capítulo 4.

Os **requisitos funcionais satisfeitos no *JaCoWeb*** em relação à auditoria (ver Tabela 1.12) são: FAU_GEN.1, FAU_GEN.2, FAU_SAR.1, FAU_SAR.2, FAU_SAR.3, FAU_SEL.1, FAU_STG.1, FAU_STG.3 e FAU_STG.4.

1.6.2 Medidas de Garantia

O TOE satisfaz os requisitos de garantia de segurança especificados em LSPP (NSA, 1999). Essa seção identifica a gerência de configuração, procedimentos de liberação (*delivery*) e operação, procedimentos de desenvolvimento do sistema, documentos guias, medidas de testes e análise de vulnerabilidades aplicadas pela equipe *JaCoWeb* para satisfazer os requisitos EAL3 do CC.

1.6.2.1 Gerência de configuração

As medidas de gerência de configuração aplicadas pela equipe *JaCoWeb* incluem a atribuição de um único identificador do produto para cada liberação do TOE. Atualmente sua identificação é Esquema de Autorização *JaCoWeb DiscMand* Versão 1.0. Associado ao identificador do produto está a lista de configuração de *hardware* e *software* que compõe uma única instância do TOE. Essas medidas de gerência de configuração estão sendo registrados dentro do seguinte documento, em desenvolvimento:

- http://www.lcmi.ufsc.br/~merkle/FinalReport.html.

Os **requisitos de garantia satisfeitos no *JaCoWeb*** em relação à gerência de configuração são: ACM_CAP.3 e ACM_SCP.1.

1.6.2.2 Liberação (delivery) e operação

A documentação que descreve a Liberação e Operação do TOE mostra quais componentes são entregues com o *JaCoWeb DiscMand* Versão 1.0, guias para instalação e avisos importantes sobre instalação e configuração do TOE. Essa documentação encontra-se parcialmente apresentada no capítulo 4. Outros elementos estão em desenvolvimento e ficarão disponíveis em:

- http://www.lcmi.ufsc.br/~merkle/FinalReport.html.

Os **requisitos de garantia satisfeitos no *JaCoWeb*** em relação à liberação e operação são: ADO_DEL.1 e ADO_IGS.1.

1.6.2.3 Desenvolvimento

Os documentos sobre o desenvolvimento do TOE são fornecidos pelos capítulos 4 e 5 que descrevem o Esquema de Autorização proposto, o projeto e a implementação da política discricionária do esquema *JaCoWeb DiscMand* 1.0. A especificação funcional do TOE, o projeto de alto nível e a modelagem das políticas de segurança são também descritos nos capítulos anteriormente citados.

A demonstração da correpondência informal, que compreende a demonstração de que todas as funções de segurança do sistema preenchem os requisitos funcionais estabelecidos no *JaCoWeb-ST*, é justificada pelo próprio modelo de segurança do CORBA, pelos componentes de *software iSaSiLk* (GRAZ, 1999a), *JacORB* (BROSE, 1997), certificados X.509 (ITU-TX509, 1993) e pelo modelo de segurança Java que fornecem as funcionalidades de segurança definidas pelos

componentes funcionais do TOE. Os objetos de serviço do CORBAsec utilizados, possuem interfaces e funcionalidades definidas em (OMG, 2000a).

Os **requisitos de garantia satisfeitos no *JaCoWeb*** em relação ao desenvolvimento são: ADV_FSP.1, ADV_HLD.2, ADV_RCR.1 e ADV_SMP.1.

1.6.2.4 Documentos Guias

Os documentos guias do administrador e do usuário estão disponíveis em

- http://www.lcmi.ufsc.br/~merkle/FinalReport.html.

Esses documentos conduzem o administrador da aplicação desenvolvida sobre os passos a serem seguidos para configurar o TOE. Igualmente os capítulos 4 e 5 desse texto indicam todas as tarefas de responsabilidade do administrador para que o TOE funcione corretamente. O usuário recebe informações sobre a aplicação desenvolvida no capítulo 4 que explica como o TOE opera, quais suas funções e operações disponíveis. Funções relacionadas ao serviço de política *PoliCap* bem como às políticas obrigatórias ainda não encontram-se disponíveis e são objetos de implementações futuras do TOE.

Os **requisitos de garantia satisfeitos no *JaCoWeb*** em relação aos documentos guias são: AGD_ADM.1 e AGD_USR.1.

1.6.2.5 Suporte ao Ciclo de Vida

A classe de requisitos de garantia que fornecem suporte ao ciclo de vida (classe ALC), define requisitos para garantir a adoção de um modelo de ciclo de vida bem definido para todos os passos do desenvolvimento do TOE, incluindo o correto uso de ferramentas e técnicas e medidas de segurança para proteger o ambiente de desenvolvimento (ISO/IEC 15408-3, 1999).

As medidas adotadas para fornecer suporte ao ciclo de desenvolvimento do TOE foram: estabelecer o uso de uma ferramenta CASE designada *VisualAge for Java* (IBM, 1999) para facilitar a localização e proteção do projeto desenvolvido, estabelecendo uma rotina e centralizando os componentes do *software* e *hardware* utilizados (máquinas *cerf.lcmi.ufsc.br* e *kernighan.lcmi.ufsc.br*). *Backups* diários de modificações foram feitos durante o processo de desenvolvimento. Nenhuma máquina externa ao domínio teve acesso ao TOE em desenvolvimento, neste período.

Os **requisitos de garantia satisfeitos no *JaCoWeb*** em relação ao suporte ao ciclo de vida são: ALC_DVS.1.

1.6.2.6 Testes

A equipe de desenvolvimento do *JaCoWeb* executou extensivos testes do TOE. Os testes executados incluem ambos testes funcionais, de cobertura (os testes demonstram a correspondência entre documentação e as funções de segurança) e profundidade (os testes são suficientes para demonstrar que as funções de segurança operam de acordo com o projeto de alto nível do TOE), para garantir que o TOE atende seus objetivos de projeto. A metodologia adotada no processo de testes foi explicada na seção 4.4.5. Os documentos relativos aos testes realizados estão disponíveis no seguinte local:

- http://www.lcmi.ufsc.br/~merkle/FinalReport.html.

Os **requisitos de garantia satisfeitos no *JaCoWeb*** em relação aos testes são: ATE_COV.2, ATE_DPT.1, ATE_FUN.1 e ATE_IND.2.

1.6.2.7 Declaração de Vulnerabilidade

Como parte do processo de teste e projeto, a equipe *JaCoWeb* executou a Análise de Vulnerabilidades óbvias[46] do *JaCoWeb DiscMand 1.0*. O objetivo dessa análise foi identificar fraquezas óbvias que poderiam ser exploradas para um possível ataque. Neste sentido, o próprio modelo CORBAsec prevê pontos chaves que podem ser aproveitados para tentativas de ataques e também lista os serviços que o modelo CORBAsec fornece para preveni-los (CHIZMADIA, 2000). Os pontos de ameaças presentes no modelo CORBAsec foram definidos na seção 3.4.9.

Para cada um dos pontos de ataque considerados no modelo CORBAsec, o projeto *JaCoWeb* possui pelo menos um mecanismo de segurança para impedir a ocorrência desses ataques, como pode ser visto na Tabela 6.5. Pontanto, o projeto *JaCoWeb* não apresenta *vulnerabilidades óbvias* no ambiente.

A declaração e análise de vulnerabilidades do projeto *JaCoWeb* está disponível no seguinte local:

- http://www.lcmi.ufsc.br/~merkle/FinalReport.html.

[46] **Vulnerabilidades ou fraquezas óbvias** são aquelas que possibilitam a exploração do ambiente com um mínimo: de entendimento do TOE, de habilidades, de sofisticação técnica e de recursos. Essas vulnerabilidades podem ser sugeridas na descrição do TOE. Vulnerabilidades óbvias incluem aquelas de conhecimento público, ou aquelas conhecidas pelo desenvolvedor ou disponíveis a partir de uma autoridade de avaliação.

Os **requisitos de garantia satisfeitos no *JaCoWeb*** em relação à declaração de vulnerabilidades são: AVA_MSU.1, AVA_SOF.1 e AVA_VLA.1.

1.7 PP Claims

Esta seção fornece a declaração de conformidade ao PP utilizado.

1.7.1 Referência ao PP

Esse TOE segue conformidade ao seguinte PP:

- *Labeled Security Protection Profile* (*LSPP*), Version 1.b, 8 October 1999, registrado pela Agência de Segurança Nacional dos Estados Unidos (NSA – *National Security Agency*) – (http://www.radium.ncsc.mil/tpep/library/protection_profiles/index.html) (NSA, 1999).

1.7.2 Refinamentos e Adições ao PP

Os seguintes requisitos funcionais e suposições foram adicionados ao PP para esse *Security Target*:

a) FCS_CKM.3 Acesso a chaves criptográficas

b) FCS_CKM.4 Destruição de chaves criptográficas

c) FCS_COP.1 Operação Criptográfica

A seguinte política de segurança organizacional foi adicionada ao PP:

P.DAC Política de Controle de Acesso Discricionária

1.8 *Rationale* (Interpretação)

Esta seção ilustra a completitude (*completeness*) e consistência desse ST. Fornece a interpretação para a seleção, criação e uso das políticas de segurança, objetivos e componentes. Os objetivos de segurança para a tecnologia de informação e para o ambiente são explicados em termos de ameaças evitadas e hipóteses assumidas. Os requisitos funcionais e de garantia são explicados em termos de objetivos almejados pelos requisitos. É feita uma análise que faz o mapeamento dos objetivos de segurança aos componentes utilizados.

Além de fornecer essa interpretação, a seção 1.6 fornece as explicações necessárias para o entendimento de como o TOE deve tratar todos os objetivos de segurança definidos.

1.8.1 Rationale *dos Objetivos de Segurança*

Essa seção fornece a interpretação para a existência de cada ameaça, declaração de política, objetivo de segurança e componentes que formam o PP.

1.8.1.1 Ameaças

Ameças tratadas pelo TOE estão descritas abaixo.

Nome	Descrição
T.ACCESS	Um usuário autorizado do TOE acessa informações ou recursos sem ter permissão do proprietário ou responsável pela informação ou recurso.
T.CAPTURE	Um invasor pode observar ou capturar dados sendo transmitidos pela rede.
T.INTEGRITY	A integridade da informação pode ser comprometida devido a erros do usuário, de transmissão ou falhas no *hardware*.
T.SECRET	Um usuário autorizado do TOE pode, intencional ou acidentalmente, observar informações armazenadas no TOE para as quais não possui autorização.
T.IMPERSON	Um invasor (interno ou externo) pode obter acesso autorizado às informações ou recursos se fazendo passar por um usuário autorizado do TOE.

A ameaça tratada pelo ambiente operacional é descrita abaixo.

Nome	Descrição
T.TUSAGE	O TOE pode ser inadvertidamente configurado, usado e administrado de forma insegura por pessoas autorizadas ou não autorizadas.

1.8.1.2 Objetivos de Segurança da Tecnologia de Informação

Os objetivos de segurança são descritos abaixo:

Nome	Descrição das Ameaças Impedidas
O.AUTHORIZATION	Esse objetivo de segurança é necessário para conter a seguinte ameaça: T.ACCESS, pois requer que o usuário seja autorizado para acessar o TOE.
O.DISCRETIONARY_ ACCESS	Esse objetivo de segurança é necessário para conter as seguintes ameaças: T.ACCESS e T.IMPERSON, pois requer que o TOE aplique a política de controle de acesso discricionária para controlar o acesso aos recursos.
O.MANDATORY_ ACCESS	Esse objetivo de segurança é necessário para conter as seguintes ameaças: T.ACCESS, T.SECRET e T.IMPERSON, pois requer que a política de controle de acesso obrigatória seja aplicada a cada acesso ao recursos do TOE.
O.AUDITING	Esse objetivo de segurança é necessário para conter as seguintes ameaças: T.ACCESS, T.CAPTURE, T.INTEGRITY, T.SECRET e T.IMPERSON, pois registra as ações de segurança relevantes dos usuários do TOE e revela essas informações apenas para administradores autorizados pela aplicação das funções de segurança.
O.RESIDUAL_ INFORMATION	Esse objetivo de segurança é necessário para conter as seguintes ameaças: T.CAPTURE e T.SECRET, pois assegura que quaisquer informações contidas em um recurso protegido não seja liberada quando o recurso é reutilizado.
O.MANAGE	Esse objetivo de segurança é necessário para conter as seguintes ameaças: T.ACCESS e T.SECRET, pois requer que apenas administradores tenham acesso à informações de gerenciamento da segurança do TOE.
O.ENFORCEMENT	Esse objetivo de segurança é necessário para conter as seguintes ameaças: T.ACCESS, T.CAPTURE, T.INTEGRITY, T.SECRET e T.IMPERSON, pois requer que o TOE aplique as políticas de segurança organizacionais no ambiente.

Na Tabela 1.19 é feito o mapeamento das ameaças e objetivos de segurança da tecnologia de informação.

	T.ACCESS	T.CAPTURE	T.INTEGRITY	T.SECRET	T.IMPERSON
O.AUTHORIZATION	X				
O.DISCRETIONARY_ ACCESS	X				X
O.MANDATORY_ ACCESS	X			X	X
O.AUDITING	X	X	X	X	X
O.RESIDUAL_ INFORMATION		X		X	
O.MANAGE	X			X	
O.ENFORCEMENT	X	X	X	X	X

Tabela 1.19 – Mapeamento das ameaças para os objetivos de segurança da tecnologia de informação (NSA, 1999).

1.8.1.3 Objetivos de Segurança do Ambiente

Os objetivos de segurança do ambiente são descritos abaixo:

Nome	Descrição das Ameaças Impedidas
O.INSTALL	Esse objetivo de segurança é necessário para conter a seguinte ameaça: T.TUSAGE, pois requer que o TOE seja entregue, instalado, gerenciado e operado de maneira que se mantenham os objetivos de segurança da IT.
O.PHYSICAL	Esse objetivo de Segurança é necessário para conter a seguinte ameaça: T.TUSAGE, pois requer que os responsáveis pelo TOE devem garantir que suas partes críticas para a política de segurança são protegidas de ataques físicos que possam comprometer os objetivos de segurança da IT.

Na Tabela 1.20 é feito o mapeamento das ameaças/suposições e objetivos de segurança da tecnologia de informação.

	T.TUSAGE	A.LOCATE	A.PROTECT	A.MANAGE	A.SENSITIVITY
O.INSTALL	X			X	X
O.PHYSICAL	X	X	X		

Tabela 1.20 – Mapeamento das ameaças/suposições para os objetivos de segurança do ambiente.

1.8.1.4 Políticas de Segurança Organizacionais

Esta seção fornece evidências demonstrando que as políticas de segurança organizacionais são tratadas/cobertas por ambos objetivos de segurança da tecnologia de informação e do ambiente. A Tabela 1.21 mostra o mapeamento entre objetivos e políticas.

Política de Segurança Organizacional	Objetivos de Segurança
P.AUTHORIZED_USERS	O.AUTHORIZATION O.MANAGE O.ENFORCEMENT
P.NEED_TO_KNOW	O.DISCRETIONARY_ACCESS O.RESIDUAL_INFORMATION O.MANAGE O.ENFORCEMENT
P.ACCOUNTABILITY	O.AUDITING O.MANAGE O.ENFORCEMENT
P.CLASSIFICATION	O.MANDATORY_ACCESS O.RESIDUAL_INFORMATION O.MANAGE O.ENFORCEMENT

Tabela 1.21 – Mapeamento das políticas de segurança organizacionais para os objetivos de segurança (NSA, 1999).

A seguinte discussão fornece detalhes sobre a cobertura de cada declaração de política de segurança organizacional (NSA, 1999).

P.AUTHORIZED_USERS

Apenas usuários autorizados a acessar informações do sistema podem acessar o sistema.

Essa política é implementada pelo objetivo O.AUTHORIZATION. O objetivo O.MANAGE suporta essa política pois requer que administradores autorizados sejam capazes de gerenciar as funções e o objetivo O.ENFORCEMENT garante que as funções são invocadas e operadas corretamente.

P.NEED_TO_KNOW

O sistema deve limitar o acesso, a modificação e a destruição de informações em recursos protegidos aos usuários autorizados que possuem uma informação para acesso (need to know).

Essa política é implementada pelo objetivo O.DISCRETIONARY_ACCESS. O objetivo O.RESIDUAL_INFORMATION garante que a informação não será fornecida a usuários que não tem nenhuma informação de acesso, quando os recursos são reutilizados. O objetivo O.MANAGE suporta essa política pois requer que administradores autorizados sejam capazes de gerenciar as funções e o objetivo O.ENFORCEMENT garante que as funções são invocadas e operadas corretamente.

P.ACCOUNTABILITY

Os usuários do sistema devem ser responsabilizados por suas ações dentro do sistema.

Essa política é implementada pelo objetivo O.AUDITING pois requer que as ações sejam registradas em um *audit trail* (registro de auditoria). O objetivo O.MANAGE suporta essa política pois requer que administradores autorizados sejam capazes de gerenciar as funções e o objetivo O.ENFORCEMENT garante que as funções são invocadas e operadas corretamente.

P.CLASSIFICATION

O sistema deve limitar o acesso a informações baseado na sensibilidade, representada por um rótulo, da informação contida nos objetos, e na habilitação formal dos usuários, representada pelos sujeitos, para acessar a informação. A aplicação das regras de acesso previnem o sujeito de acessar informações de maior sensibilidade e previnem o sujeito de causar o rebaixamento (downgrading) das informações para um nível de sensibilidade menor.

Essa política é implementada pelo objetivo O.MANDATORY_ACCESS. O objetivo O.RESIDUAL_INFORMATION garante que a informação não será fornecida a usuários que não tem habilitação de acesso, quando os recursos são reutilizados. O objetivo O.MANAGE suporta essa política pois requer que administradores autorizados sejam capazes de gerenciar as funções e o objetivo O.ENFORCEMENT garante que as funções são invocadas e operadas corretamente.

1.8.2 Rationale *dos Requisitos de Segurança*

Esta seção fornece evidências para suportar a consistência e completitude dos componentes funcionais e de garantia que compreendem o LSPP (NSA, 1999).

Os componentes funcionais selecionados fornecem cobertura completa dos objetivos de segurança definidos. O mapeamento entre componentes e objetivos de segurança é representado na Tabela 1.22.

Objetivo de Segurança	Componente Funcional
O.AUTHORIZATION	1.6.1.2 Definição dos Atributos dos Usuários (FIA_ATD.1) 1.6.1.2 Força (*Strength*) dos Dados de Autenticação (FIA_SOS.1) 1.6.1.2 Autenticação (FIA_UAU.1) 1.6.1.2 *Feedback* da Autenticação Protegida (FIA_UAU.7) 1.6.1.2 Identificação (FIA_UID.1) 1.6.1.1 Gerência dos Dados de Autenticação (FMT_MTD.1 (4))
O.DISCRETIONARY_ ACCESS	1.6.1.3 Política de Controle de Acesso Discricionária (FDP_ACC.1) 1.6.1.3 Funções de Controle de Acesso Discricionárias (FDP_ACF.1) 1.6.1.2 Definição dos Atributos dos Usuários (FIA_ATD.1) 1.6.1.2 Ligação (*binding*) entre o usuário e sujeito (FIA_USB.1) 1.6.1.1 Gerência dos Atributos de Segurança do Objeto (FMT_MSA.1) 1.6.1.1 Inicialização dos Atributos Estáticos (FMT_MSA.3) 1.6.1.1 Revogação de Atributos de Objeto (FMT_REV.1 (2))
O.MANDATORY_ ACCESS	1.6.1.3 Exportação de Dados de Usuário Não Rotulados (FDP_ETC.1) 1.6.1.3 Exportação de Dados de Usuário Rotulados (FDP_ETC.2) 1.6.1.3 Política de Controle de Acesso Obrigatória (FDP_IFC.1) 1.6.1.3 Funções de Controle de Acesso Obrigatórios (FDP_IFF.2) 1.6.1.3 Importação de Dados de Usuário Não-Rotulados (FDP_ITC.1) 1.6.1.3 Importação de Dados de Usuário Rotulados (FDP_ITC.2) 1.6.1.2 Ligação (*binding*) entre o usuário e sujeito (FIA_USB.1) 1.6.1.1 Gerência dos Atributos de Segurança do Objeto (FMT_MSA.1) 1.6.1.1 Inicialização dos Atributos Estáticos (FMT_MSA.3)
O.AUDITING	1.6.1.6 Geração de Dados de Auditoria (FAU_GEN.1) 1.6.1.6 Associação da Identidade do Usuário (FAU_GEN.2) 1.6.1.6 Revisão da Auditoria (FAU_SAR.1) 1.6.1.6 Revisão da Auditoria Restrita (FAU_SAR.2) 1.6.1.6 Revisão da Auditoria Selecionada (FAU_SAR.3) 1.6.1.6 Auditoria Seletiva (FAU_SEL.1) 1.6.1.6 Garantias de Disponibilidade de Dados de Auditoria (FAU_STG.1) 1.6.1.6 Ação no caso da possível perda dos Dados de Auditoria (FAU_STG.3) 1.6.1.6 Prevenção da Perda dos Dados de Auditoria (FAU_STG.4) 1.6.1.2 Ligação (*binding*) entre o usuário e sujeito (FIA_USB.1) 1.6.1.1 Gerência da trilha (*trail*) de Auditoria (FMT_MTD.1 (1)) 1.6.1.1 Gerência dos Eventos de Auditoria (FMT_MTD.1 (2)) 1.6.1.4 *Time Stamps* Confiáveis (FPT_STM.1)
O.RESIDUAL_ INFORMATION	1.6.1.3 Proteção às Informações Residuais dos Objetos (FDP_RIP.2)
O.MANAGE	1.6.1.6 Revisão da Auditoria (FAU_SAR.1) 1.6.1.6 Revisão da Auditoria Selecionada (FAU_SAR.3) 1.6.1.6 Auditoria Seletiva (FAU_SEL.1) 1.6.1.6 Ação no caso da possível perda dos Dados de Auditoria (FAU_STG.3) 1.6.1.6 Prevenção da Perda dos Dados de Auditoria (FAU_STG.4) 1.6.1.1 Gerência da trilha (*trail*) de Auditoria (FMT_MTD.1 (1)) 1.6.1.1 Gerência dos Eventos de Auditoria (FMT_MTD.1 (2)) 1.6.1.1 Gerência dos Atributos dos Usuários (FMT_MTD.1 (3)) 1.6.1.1 Gerência dos Dados de Autenticação (FMT_MTD.1 (4)) 1.6.1.1 Revogação de Atributos de Usuário (FMT_REV.1 (1)) 1.6.1.1 Papéis (*roles*) na gerência da segurança (FMT_SMR.1) 1.6.1.5 Acesso a chaves criptográficas (FCS_CKM.3) 1.6.1.5 Destruição de chaves criptográficas (FCS_CKM.4) 1.6.1.5 Operação criptográfica (FCS_COP.1)
O.ENFORCEMENT	1.6.1.4 Teste da Máquina Abstrata (FPT_AMT.1) 1.6.1.4 Mediação de Referência (FPT_RVM.1) 1.6.1.4 Separação de Domínios (FPT_SEP.1) 1.6.1.5 Acesso a chaves criptográficas (FCS_CKM.3) 1.6.1.5 Destruição de chaves criptográficas (FCS_CKM.4) 1.6.1.5 Operação criptográfica (FCS_COP.1)

Tabela 1.22 – Mapeamento entre objetivos de segurança e componentes funcionais (NSA, 1999).

A discussão seguinte fornece detalhes sobre as evidências de cobertura de cada um dos objetivos de segurança.

O_AUTHORIZATION

As TSF's devem garantir que apenas usuários autorizados tenham acesso ao TOE e aos seus recursos.

Usuários autorizados a acessar o TOE são definidos usando um processo de identificação e autenticação [seção 1.6.1.2]. Para garantir acesso autorizado ao TOE, dados de autenticação são protegidos [seção 1.6.1.2, seção 1.6.1.1]. A força do mecanismo de autenticação deve ser suficiente para garantir que usuários não autorizados não possam se fazer passar por usuários autorizados [seção 1.6.1.2].

O_DISCRETIONARY_ACCESS

As TSF's devem controlar acesso aos recursos baseado na identidade dos usuários. As TSF's devem permitir aos usuários autorizados especificar quais recursos podem ser acessados por quais usuários.

O controle de acesso discricionário deve ter um escopo de controle definido [seção 1.6.1.3]. As regras da política discricionária devem ser definidas [seção 1.6.1.3]. Os atributos de segurança dos objetos usados para garantir a aplicação da política discricionária devem também ser definidos. Os atributos de segurança dos sujeitos usados para garantir a aplicação da política discricionária são também determinados [seção 1.6.1.2]. Usuários autorizados devem ser capazes de controlar quem tem acesso aos seus objetos e capazes de revogar estes direitos de acesso aos seus objetos [seção 1.6.1.1]. A proteção dos objetos deve ser contínua, iniciando durante a sua criação [seção 1.6.1.1].

O_MANDATORY_ACCESS

As TSF's devem controlar acesso aos recursos baseado na sensibilidade e categorias das informações que estão sendo acessadas e na habilitação do sujeito que está tentando ter acesso à informação.

Atributos e regras para o controle de acesso obrigatório devem ser definidos e devem ter um escopo de controle definido [seção 1.6.1.3]. As regras para importar dados não-rotulados e dados rotulados devem ser cobertas, bem como a exportação de dados não-rotulados e dados rotulados [seção 1.6.1.3]. Finalmente, se a política de controle de acesso obrigatória deve ser corretamente aplicada, é requerido que atributos estáticos suficientes e corretos sejam associados

com cada objeto [seção 1.6.1.1], e que a ligação entre processos e atributos do usuário para o qual os processos operam sejam corretos e impossíveis de serem forjados [seção 1.6.1.2].

O.AUDITING

As TSF's devem registrar as ações de segurança relevantes dos usuários do TOE. As TSF's devem apresentar essa informação para usuários autorizados.

As ações de segurança relevantes devem ser definidas, passíveis de auditoria e capazes de serem associadas com os usuários individuais [seção 1.6.1.6, seção 1.6.1.2]. A trilha de auditoria (*audit trail*) deve ser protegida de forma que apenas usuários autorizados possam acessá-la [seção 1.6.1.6]. As TSF's devem fornecer a capacidade de verificar as ações de um usuário individual [seção 1.6.1.6, seção 1.6.1.2]. A trilha de auditoria deve ser completa. Os *time stamps* associados devem ser confiáveis [seção 1.6.1.6]. Um administrador autorizado deve ser capaz de revisar e gerenciar a trilha de auditoria [seção 1.6.1.6, seção 1.6.1.1].

O.RESIDUAL_INFORMATION

As TSF's devem garantir que quaisquer informações contidas em um recurso protegido não seja liberada quando o recurso é reutilizado.

Informações residuais (prévias/permanentes) associadas com os objetos definidos em um TOE devem ser eliminadas a fim de reusar o objeto que contém essa informação [seção 1.6.1.3].

O.MANAGE

As TSF's devem fornecer todas as funções e facilidades necessárias para suportar administradores autorizados que são responsáveis pela gerência da segurança do TOE.

As TSF's devem fornecer a um administrador autorizado formas de gerenciar o TOE [seção 1.6.1.1, seção 1.6.1.5]. O administrador deve ser capaz de administrar sessões de usuário [seção 1.6.1.1]. O administrador deve ser capaz de revisar e gerenciar a trilha de auditoria [seção 1.6.1.1, seção 1.6.1.6].

O.ENFORCEMENT

As TSF's devem ser registradas e implementadas de forma a garantir que as políticas de segurança organizacionais são aplicadas no ambiente.

As TSF's devem elaborar e garantir as decisões das TSP's [seção 1.6.1.4]. Elas devem ser protegidas das interferências que poderiam impedir a execução de suas funções [seção 1.6.1.4, seção 1.6.1.5]. Adicionalmente, o TOE deve fornecer a capacidade de demonstrar a operação

correta da máquina abstrata subjacente das TSF's [seção 1.6.1.4]. A correção desse objetivo é garantida pelos requisitos de garantia [seção 1.6.2].

Esse objetivo fornece suporte global aos outros objetivos de segurança do TOE, protegendo as partes do TOE que implementa as políticas e garante a aplicação das políticas de segurança.

1.8.2.1 Dependências

A Tabela 1.23 mostra as dependências que existem. Um 'X' indica que a dependência foi satisfeita. Um 'O' indica uma dependência opcional onde uma das opções foi satisfeita.

Seção	Identificador do CC	FAU_GEN.1	FAU_SAR.1	FAU_STG.1	FDP_ACC.1	FDP_ACF.1	FDP_IFC.1	FDP_IFF.1	FDP_ITC.1	FIA_ATD.1	FIA_UAU.1	FIA_UID.1	FMT_MSA.1	FMT_MSA.3	FMT_MTD.1	FMT_SMR.1	FPT_STM.1
1.6.1.6	FAU_GEN.1																X
1.6.1.6	FAU_GEN.2	X										X					
1.6.1.6	FAU_SAR.1	X															
1.6.1.6	FAU_SAR.2		X														
1.6.1.6	FAU_SAR.3		X														
1.6.1.6	FAU_SEL.1	X													X		
1.6.1.6	FAU_STG.1	X															
1.6.1.6	FAU_STG.3																
1.6.1.6	FAU_STG.4			X													
1.6.1.3	FDP_ACC.1	X															
1.6.1.3	FDP_ACF.1					X											
1.6.1.3	FDP_RIP.2				X												
1.6.1.2	FIA_ATD.1																
1.6.1.2	FIA_SOS.1																
1.6.1.2	FIA_UAU.1											X					
1.6.1.2	FIA_UAU.7										X						
1.6.1.2	FIA_UID.1																
1.6.1.2	FIA_USB.1									X							
1.6.1.1	FMT_MSA.1				O		O									X	
1.6.1.1	FMT_MSA.3												X			X	
1.6.1.1	FMT_MTD.1															X	
1.6.1.1	FMT_MTD.1															X	
1.6.1.1	FMT_MTD.1															X	
1.6.1.1	FMT_MTF.1															X	
1.6.1.1	FMT_REV.1															X	
1.6.1.1	FMT_REV.1															X	
1.6.1.1	FMT_SMR.1											X					
1.6.1.4	FPT_AMT.1																
1.6.1.4	FPT_RVM.1																
1.6.1.4	FPT_SEP.1																
1.6.1.4	FPT_STM.1																
1.6.1.5	FCS_CKM.3								X								
1.6.1.5	FCS_CKM.4								X								
1.6.1.5	FCS_COP.1								X								

Tabela 1.23 – Dependências dos componentes funcionais (NSA, 1999).

1.8.2.2 Rationale *para os Requisitos de Garantia*

O nível EAL3 foi escolhido como o mais apropriado para esse ST. Esse ST foi desenvolvido para um ambiente genérico com um risco moderado dos seus recursos. Pretende-se que os produtos usados nesses ambientes sejam geralmente disponíveis, sem modificações para tratar das necessidades de segurança do ambiente (NSA, 1999).

1.8.3 Rationale *da Especificação Sumária do TOE*

Esta seção demonstra que as funções de segurança do TOE e as medidas de garantia são apropriados para tratar os requisitos de segurança almejados.

1.8.3.1 Funções de Segurança do TOE

As funções de segurança do TOE especificadas trabalham em conjunto para satisfazer os requisitos funcionais de segurança do TOE. A Tabela 1.24 fornece o mapeamento dos SFR's para as funções de segurança para mostrar que todos os SFR's são inseridos em uma função de segurança.

Função de Segurança	Requisito Funcional de Segurança
Auditoria	FAU_GEN.1
	FAU_GEN.2
	FAU_SAR.1
	FAU_SAR.2
	FAU_SAR.3
	FAU_SEL.1
	FAU_STG.1
	FAU_STG.3
	FAU_STG.4
Identificação e Autenticação	FIA_ATD.1
	FIA_SOS.1
	FIA_UAU.1
	FIA_UAU.7
	FIA_UID.1
	FIA_USB.1
Gerência da Segurança	FMT_MSA.1
	FMT_MSA.3
	FMT_MTD.1 (1)
	FMT_MTD.1 (2)
	FMT_MTD.1 (3)
	FMT_MTD.1 (4)
	FMT_REV.1 (1)
	FMT_REV.1 (2)
	FMT_SMR.1
Proteção dos Dados	FDP_ACC.1
	FDP_ACF.1
	FDP_ETC.1
	FDP_ETC.2
	FDP_IFC.1
	FDP_IFF.2
	FDP_ITC.1
	FDP_ITC.2
	FDP_RIP.2
Proteção das Funções de Segurança	FPT_AMT.1
	FPT_RVM.1
	FPT_SEP.1
	FPT_STM.1
Suporte Criptográfico	FCS_CKM.3
	FCS_CKM.4
	FCS_COP.1

Tabela 1.24 – Mapeamento das SFR's para as Funções de Segurança.

A seção 1.5.1 e a seção 1.6.1 apresentam as funções de segurança executadas e implementadas pelo TOE, explicando cada um dos componentes da Tabela 1.24.

1.8.3.2 Requisitos de Garantias do TOE

O TOE satisfaz os requisitos de garantia SAR's do LSPP (NSA, 1999). A seção 1.5.2 e a seção 1.6.2 desse documento identificam as medidas de gerência de configuração, de liberação e operação do TOE, procedimentos de desenvolvimento do sistema, documentos guias do TOE, medidas de teste e análise de vulnerabilidades aplicadas pela equipe *JaCoWeb* para satisfazer os

requisitos de garantia do nível EAL3 do CC. A Tabela 1.25 ilustra a concordância das medidas de garantia com os requisitos de garantia conforme declarados na seção 1.5.2.

	Gerência de Configuração	Liberação (*delivery*) e Operação	Desenvolvimento	Documentos Guias	Suporte ao Ciclo de Vida	Testes	Declaração de Vulnerabilidade
ACM_CAP.3	✓						
ACM_SCP.1	✓						
ADO_DEL.1		✓					
ADO_IGS.1		✓					
ADV_FSP.1			✓				
ADV_HLD.2			✓				
ADV_RCR.1			✓				
ADV_SMP.1			✓				
AGD_ADM.1				✓			
AGD_USR.1				✓			
ALC_DVS.1					✓		
ATE_COV.2						✓	
ATE_DPT.1						✓	
ATE_FUN.1						✓	
ATE_IND.2						✓	
AVA_MSU.1							✓
AVA_SOF.1							✓
AVA_VLA.1							✓

Tabela 1.25 – Matriz de conformidade com as Medidas de Garantia.

1.8.4 Rationale *do PP Claims*

Este ST está em conformidade com o LSPP (NSA, 1999) porque ele contém todos os requisitos funcionais com os apropriados refinamentos e objetivos de segurança conforme identificado no LSPP com exceções e adições descritas e justificadas em 1.7.

ANEXO 2 – AVALIAÇÃO DO *JACOWEB-ST*

2.1 Avaliação do *JaCoWeb-ST*

O objetivo da avaliação do ST é determinar se o Anexo 1, ou seja, o *JaCoWeb*-ST, é completo, consistente, tecnicamente confiável e para determinar se o *JaCoWeb-ST* fornece base adequada para a avaliação do TOE. A seqüência de tarefas executadas nesta avaliação segue as ações relacionadas na Tabela 6.11 do capítulo 6.

2.1.1 Avaliação da Descrição do TOE – ASE_DES.1

ASE_DES.1 Veredito:

A equipe de avaliação concluiu que o ST atendeu os critérios de avaliação do ASE_DES.1 do ST pois os elementos de ações do avaliador ASE_DES.1.1E, ASE_DES.1.2E e ASE_DES.1.3E foram completados com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

ASE_DES.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ASE_DES.1.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ASE_DES.1.1E Rationale:

A equipe de avaliação verificou (*checked*) e examinou (*examined*) o Anexo 1. O avaliador examinou a evidência e determinou que a Descrição do TOE na seção 1.2 descreve o produto e o escopo e limites do TOE, ambas de formas físicas e lógicas. Como resultado, o avaliador determinou que todos os requisitos para essa atividade foram satisfeitos.

ASE_DES.1.2E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ASE_DES.1.2E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ASE_DES.1.2E Rationale:

A Descrição do TOE no Anexo 1 foi examinada e considerada uma descrição consistente da funcionalidade fornecida pelo TOE. A seção 1.2.1, do Anexo 1, descreve a arquitetura do TOE, que é composto por um serviço de política *PoliCap*, applets e servidores de aplicação e o pacote *JaCoWebSecurity*. A seção 1.2.2 fornece as características de segurança do TOE. Para cada característica de segurança é fornecido um breve sumário de como o TOE trata daquela característica. A seção 6.4.2.3 fornece informações adicionais. As informações encontradas nessa seção são consistentes com a informação encontrada na Descrição do TOE. Como resultado, o avaliador determinou que todos os requisitos para essa atividade foram satisfeitos.

ASE_DES.1.3E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ASE_DES.1.3E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ASE_DES.1.3E Rationale:

No Anexo 1, os ítens que contêm material descritivo considerando o TOE são o 1.1, Introdução ao *Security Target*, o 1.2, Descrição do TOE e o 1.6, Especificação Sumária do TOE. A Descrição do TOE descreve de forma breve as características do TOE, enquanto a Especificação Sumária do TOE fornece uma definição de alto nível dessas características. Como resultado, o avaliador determinou que todos os requisitos para essa atividade foram satisfeitos.

2.1.2 Avaliação do Ambiente de Segurança - ASE_ENV.1

ASE_ENV.1 Veredito:

A equipe de avaliação concluiu que o ST atendeu os critérios de avaliação do ASE_ENV.1 do ST pois os elementos de ações do avaliador ASE_ENV.1.1E e ASE_ENV.1.2E foram completados com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

ASE_ENV.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ASE_ENV.1.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ASE_ENV.1.1E Rationale:

A equipe de avaliação verificou (*checked*) e examinou (*examined*) o Anexo 1 e o documento (NSA, 1999). O avaliador examinou as evidências e determinou que o Ambiente de Segurança explica todas as suposições, ameaças e políticas de segurança organizacionais relativas ao TOE. Como o ST declara conformidade com o PP descrito em (NSA, 1999), as suposições e ameaças foram usadas a partir desse PP. O avaliador examinou as evidências e considerou que todas as ameaças identificadas foram claramente explicadas em termos do recurso sujeito ao ataque. O avaliador examinou as evidências e considerou que as políticas de segurança organizacionais foram explicadas com detalhes suficientes para torná-las facilmente entendidas e para definir objetivos de segurança que trate de cumprir as mesmas. Como resultado, o avaliador determinou que todos os requisitos para essa atividade foram satisfeitos.

ASE_ENV.1.2E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ASE_ENV.1.2E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ASE_ENV.1.2E Rationale:

O avaliador examinou o Anexo 1 e determinou que o Ambiente de Segurança do TOE está coerente e internamente consistente. O avaliador examinou a evidência e determinou que o Ambiente de Segurança do TOE estava consistente com a Introdução ao ST e com a Descrição do TOE. Como resultado, o avaliador determinou que todos os requisitos para essa atividade foram satisfeitos.

2.1.3 Avaliação da Introdução ao ST – ASE_INT.1

ASE_INT.1 Veredito:

A equipe de avaliação concluiu que o ST atendeu os critérios de avaliação do ASE_INT.1 do ST pois os elementos de ações do avaliador ASE_INT.1.1E, ASE_INT.1.2E e ASE_INT.1.3E foram completados com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

ASE_INT.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ASE_INT.1.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ASE_INT.1.1E Rationale:

A equipe de avaliação verificou o Anexo 1. O avaliador verificou a evidência e considerou que a Introdução ao ST contém informações sobre a identificação do ST, um resumo do ST e uma declaração de conformidade. Como resultado, o avaliador determinou que todos os requisitos para essa atividade foram satisfeitos.

ASE_INT.1.2E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ASE_INT.1.2E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ASE_INT.1.2E Rationale:

A equipe de avaliação examinou a Introdução ao ST e determinou que está coerente e internamente consistente. A Introdução ao ST não contém nenhuma declaração ambígua ou vaga.

ASE_INT.1.3E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ASE_INT.1.3E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ASE_INT.1.3E Rationale:

A equipe de avaliação verificou e examinou o Anexo 1. A Introdução ao ST foi examinada e considerada consistente com o restante do Anexo 1. As declarações de conformidade aos Critérios Comuns são consistentes com as declarações encontradas na seção relativa ao PP Claims (seção 1.7). O Resumo do ST é consistente com a Descrição e a Especificação Sumária do TOE. As Convenções descritas são consistentes com aquelas usadas nos Requisitos de Segurança do TOE. Como resultado dessas atividades, o avaliador determinou que todos os requisitos para essa atividade foram satisfeitos.

2.1.4 Avaliação dos Objetivos de Segurança – ASE_OBJ.1

ASE_OBJ.1 Veredito:

A equipe de avaliação concluiu que o ST atendeu os critérios de avaliação do ASE_OBJ.1 do ST, pois os elementos de ações do avaliador ASE_OBJ.1.1E e ASE_OBJ.1.2E foram completados com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

ASE_OBJ.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ASE_OBJ.1.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ASE_OBJ.1.1E Rationale:

A equipe de avaliação verificou (*checked*) e examinou (*examined*) o Anexo 1 e o documento (NSA, 1999). O avaliador examinou as evidências e considerou definidos os Objetivos de Segurança do TOE e os Objetivos de Segurança do Ambiente. O avaliador verificou e examinou as evidências e considerou que os objetivos de segurança estão claramente definidos. O avaliador examinou as evidências e considerou que cada objetivo de segurança do TOE faz referência a uma ameaça identificada e que cada objetivo de segurança do ambiente faz referência a uma suposição considerada. Como o ST declara conformidade com o PP descrito em (NSA, 1999), o avaliador verificou que os objetivos de segurança existentes no PP foram referenciados e/ou transcritos para o ST. O avaliador examinou as evidências e determinou que a interpretação (*rationale*) dos objetivos de segurança identificam as ameaças à segurança e que os objetivos de segurança são apropriados para conter essas ameaças. Como resultado dessas atividades, o avaliador determinou que todos os requisitos para essa atividade foram satisfeitos.

ASE_OBJ.1.2E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ASE_OBJ.1.2E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ASE_OBJ.1.2E Rationale:

A equipe de avaliação examinou o Anexo 1 e considerou que os Objetivos de Segurança estão completos, coerentes e internamente consistentes. Cada objetivo de segurança, para ambos TOE e ambiente, é definido de forma que um mapeamento entre as suposições e ameaças é possível. Pelo menos um dos objetivos de segurança é mapeado a pelo menos uma ameaça ou suposição. Todas as suposições e ameaças identificadas são satisfeitas sem contradição. Na seção 1.8.1, que descreve a interpretação (*rationale*) dos objetivos de segurança, objetivos de segurança da Tecnologia de Informação e do Ambiente são explicados, de forma que se tem um mapeamento claro, juntamente com uma declaração de como o objetivo impede cada ameaça associada. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

2.1.5 Avaliação do PP Claims – ASE_PPC.1

ASE_PPC.1 Veredito:

A equipe de avaliação concluiu que o ST atendeu os critérios de avaliação do ASE_PPC.1 do ST pois os elementos de ações do avaliador ASE_PPC.1.1E e ASE_PPC.1.2E foram completados com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

ASE_PPC.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ASE_PPC.1.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ASE_PPC.1.1E Rationale:

A equipe de avaliação verificou (*checked*) e examinou (*examined*) o Anexo 1 e o documento (NSA, 1999). O avaliador verificou as evidências e determinou que elas descrevem cada conformidade a algum PP, identificam os requisitos de segurança do TOE que satisfazem as operações permitidas do PP e identifica os objetivos de segurança e requisitos de segurança da tecnologia de informação que foram adicionados. Como resultado dessas atividades, o avaliador determinou que todos os requisitos para essa atividade foram satisfeitos.

ASE_PPC.1.2E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ASE_PPC.1.2E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ASE_PPC.1.2E Rationale:

A equipe de avaliação examinou o Anexo 1 e o documento (NSA, 1999). O avaliador examinou as evidências e verificou que cada requisito do PP foi declarado novamente no ST. O avaliador verificou o mapeamento de todos os requisitos do PP para um requisito correspondente no ST. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

2.1.6 Avaliação dos Requisitos de Segurança da IT - ASE_REQ.1

ASE_REQ.1 Veredito:

A equipe de avaliação concluiu que o ST atendeu os critérios de avaliação do ASE_REQ.1 do ST pois os elementos de ações do avaliador ASE_REQ.1.1E e ASE_REQ.1.2E foram completados com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

ASE_REQ.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ASE_REQ.1.1E. Portanto, o veredito CORRETO (*PASS*) foi emitido para esse elemento de ação do avaliador.

ASE_REQ.1.1E Rationale:

A equipe de avaliação verificou (*checked*) e examinou (*examined*) o Anexo 1. O avaliador verificou as evidências e considerou que os Requisitos Funcionais de Segurança do TOE estão claramente identificados como requisitos funcionais de segurança do TOE e foram retirados da parte 2 (ISO/IEC 15408-2, 1999) do CC que descreve os componentes de requisitos funcionais. O avaliador verificou as evidências e considerou que cada componente funcional de segurança e cada componente de garantia foram corretamente transcritos para o ST. O evaliador verificou as evidências e considerou que os requisitos de garantia de segurança do TOE estão claramente identificados como requisitos de garantia de segurança do TOE e foram retirados da parte 3 (ISO/IEC 15408-3, 1999) do CC que descreve os componentes de requisitos de garantia. O avaliador verificou as evidências e considerou que a declaração dos requisitos de garantia de segurança do TOE compõe o nível EAL 3 e que não existem requisitos de garantia não incluídos no nível EAL 3. O avaliador verificou as evidências e considerou que todas as operações dos requisitos de segurança da tecnologia de informação foram identificados e executados corretamente. O avaliador examinou a interpretação dos requisitos de segurança e determinou que todas as dependências requeridas pelos requisitos de segurança da tecnologia de informação são explicados e satisfeitos pelo

Anexo 1. O avaliador verificou e considerou que o ST inclui 'força' criptográfica mínima (*SOF*) e que identifica os requisitos funcionais. O avaliador examinou as evidências e determinou que a interpretação dos requisitos de segurança é consistente com os objetivos de segurança para o TOE. O avaliador examinou a interpretação dos requisitos de segurança e determinou que os requisitos de segurança da tecnologia de informação para o TOE são adequados para tratar de todos os objetivos de segurança definidos. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

ASE_REQ.1.2E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ASE_REQ.1.2E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ASE_REQ.1.2E Rationale:

A equipe de avaliação examinou os requisitos de segurança da tecnologia de informação e a interpretação dos requisitos de segurança, e determinou que essas seções estão completas, coerentes e internamente consistentes. Uma análise independente foi realizada na unidade de trabalho ASE_REQ.1-3 que mostra que todas as operações nos requisitos de segurança da tecnologia de informação foram corretamente executadas. Uma análise independente foi executada na unidade de trabalho ASE_REQ.1-19, que mostra que os requisitos de segurança garantem a satisfação de todos os objetivos de segurança do TOE. O avaliador determinou que nenhum requisito de segurança entra em conflito com qualquer outro requisito de segurança. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

2.1.7 Avaliação do Requisitos de Segurança explicitamente definidos na IT - ASE_SRE.1

ASE_SRE.1 Veredito:

A equipe de avaliação concluiu que o ST atendeu os critérios de avaliação do ASE_SRE.1 do ST pois os elementos de ações do avaliador ASE_SRE.1.1E e ASE_SRE.1.2E foram completados com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

ASE_SRE.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ASE_SRE.1.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ASE_SRE.1.1E Rationale:

O Anexo 1 não contém requisitos de segurança explicitamente definidos. As unidades de trabalho para esse elemento de ação do avaliador estão trivialmente satisfeitas.

ASE_SRE.1.2E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ASE_SRE.1.2E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ASE_SRE.1.2E Rationale:

O Anexo 1 não contém requisitos de segurança explicitamente definidos. As unidades de trabalho para esse elemento de ação do avaliador estão trivialmente satisfeitas.

2.1.8 Avaliação da Especificação Sumária do TOE - ASE_TSS.1

ASE_TSS.1 Veredito:

A equipe de avaliação concluiu que o ST atendeu os critérios de avaliação do ASE_TSS.1 do ST pois os elementos de ações do avaliador ASE_TSS.1.1E e ASE_TSS.1.2E foram completados com sucesso. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse componente de garantia.

ASE_TSS.1.1E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ASE_TSS.1.1E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ASE_TSS.1.1E Rationale:

A equipe de avaliação verificou (*checked*) e examinou (*examined*) o Anexo 1. O avaliador examinou as evidências e considerou que a Especificação Sumária do TOE descreve as funções de segurança da tecnologia de informação e as medidas de garantia do TOE. O avaliador verificou as evidências e considerou que é fornecido um mapeamento das funções de segurança da tecnologia de informação para os requisitos funcionais de segurança do TOE e, também, que cada função de segurança da tecnologia de informação é mapeada a pelo menos um requisito funcional de segurança do TOE. O avaliador examinou as evidências e determinou que cada função de segurança da tecnologia de informação é apresentada de forma que pode se ter um entendimento claro do objetivo da função e todas as referências aos mecanismos de segurança são relacionados às funções de segurança da tecnologia de informação. O avaliador examinou as evidências e determinou que a interpretação da especificação sumária do TOE demonstra que as funções de segurança da tecnologia de informação são adequadas para tratar os requisitos funcionais de segurança do TOE e que a combinação das funções de segurança da tecnologia de informação especificadas trabalham em conjunto para satisfazer os requisitos funcionais de segurança do TOE. O avaliador examinou as evidências e determinou que as medidas de garantia tratam os requisitos de garantia de segurança do TOE. Como resultado dessas atividades, o avaliador determinou que todos os requisitos para essa atividade foram satisfeitos.

ASE_TSS.1.2E Veredito:

A equipe de avaliação concluiu com sucesso o elemento de ação ASE_TSS.1.2E. Portanto, o veredito **CORRETO (*PASS*)** foi emitido para esse elemento de ação do avaliador.

ASE_TSS.1.2E Rationale:

A equipe de avaliação examinou a Especificação Sumária do TOE e a Interpretação da Especificação Sumária do TOE no Anexo 1, e determinou que essas duas seções estão completas, coerentes e internamente consistentes. As funções de segurança da tecnologia de informação e as medidas de garantia são mutuamente cooperativas para atingir os requisitos de segurança do TOE. Cada função de segurança da tecnologia de informação é mapeada para um requisito funcional de segurança e cada medida de garantia é mapeada para um requisito de garantia de segurança do TOE. Uma análise independente também foi executada

nas unidades de trabalho ASE_TSS.1-6 e ASE_TSS.1-9 que mostram esses mapeamentos. A 'força' das funções é consistente tanto nas funções de segurança da tecnologia de informação quanto nos requisitos de segurança da tecnologia de informação. As funções de segurança da tecnologia de informação e as medidas de garantia são suficientes para garantir que todos os requisitos de segurança do TOE especificados sejam satisfeitos. Ambas a especificação sumária do TOE e sua interpretação foram consideradas internamente consistentes. Como resultado dessas atividades, o avaliador determinou que os requisitos para essa atividade foram satisfeitos.

REFERÊNCIAS BIBLIOGRÁFICAS

[Abadi e Needham 1996] Martin Abadi e Roger Needham. Prudent Engineering Practice for Cryptographic Protocols. *IEEE Transactions on Software Engineering*, v. 22, n. 1, p. 6-15, 1996.

[Abrams 1998] Marc Abrams. *WWW Beyond the Basics*, Prentice-Hall, ISBN 0-13-954785-1, 1998. (http://ei.cs.vt.edu/~wwwbtb/book/index.html)

[Adams 1996] C. Adams. The Simple Public-Key GSS-API Mechanism (SPKM). *Internet Draft RFC 2025*, October 1996. (http://www.ietf.org/rfc/rfc2025.txt)

[Adiron 2000] Adiron Inc. *ORBASEC SL2 User Guide – Version 2.1.4*. Adiron LLC CASE Center, Syracuse University, NY, July 2000 (http://www.adiron.com/).

[Adiron 1999] Adiron Inc. *Control – Access Control for ORBAsec SL2 Version 2.1*, Adiron LLC CASE Center, Syracuse University, NY, December 1999 (http://www.adiron.com/).

[Alireza, Lang, Padelis, *et al.* 2000] A. Alireza, U. Lang, M. Padelis, R. Schreiner and M. Schumacher. The Challenges of CORBA Security, *In: Workshop Sicherheit in Mediendaten*, June 2000, to appear, published by Springer.
(http://www.cl.cam.ac.uk/users/ul201/research.html)

[Amoroso 1994] Edward Amoroso. *Fundamentals of Computer Security*. Englewood Cliffs, NJ:Prentice-Hall, 1994.

[Bacon, Moody, Bates, *et al.* 2000] Jean Bacon, Ken Moody, John Bates, Richard Hayton, Chaoying Ma, Andrew McNeil, Oliver Seidel and Mark Spiteri. Generic Support for Distributed Applications. *IEEE Computer*, v. 33, n. 3, p.68-76, March 2000.

[Bell 1986] D. Elliot Bell. Secure Computer Systems: A Network Interpretation. *In: Proceedings of the 2nd Annual Computer Security Application Conference*, p. 32-39, Vienna, Virginia, USA, 1986 (http://www.acsac.org/pastconf/acsac02.html).

[Bell e LaPadula 1976] D. Elliot Bell e Leonard J. LaPadula. Security Computer Systems: Unified Exposition and Multics Interpretation. *MITRE Technical Report MTR-2297 Rev. 1*, MITRE Corporation, Bedford, MA, March 1976.

[Belvin 2000] Frank Belvin. CEM: Introduction and Overview. Transparências *In: ICCC First International Common Criteria Conference*, 23-25 May 2000, Baltimore, Maryland, U.S.A. (http://niap.nist.gov/cc-scheme/iccc/trackd.html)

[Beznosov e Deng 1999] Konstantin Beznosov and Yi Deng. A Framework for Implementing Role-based Access Control Using CORBA Security Service. *In: Proceedings of 4rd ACM Workshop on Role-Based Access Control*, p. 19-30, October, 1999. (http://www.acm.org/pubs/contents/proceedings/commsec/319171/)

[Biba 1977] Kenneth J. Biba. Integrity Considerations for Secure Computer Systems. *MITRE Technical Report MTR-3153*, MITRE Corporation, Bedford, MA, April 1977.

[Blakley 1999] Bob Blakley. *CORBA Security: An Introduction to Safe Computing with Objects*, The Addison-Wesley Object Technology Series, October 1999.

[Blakley 1997] B. Blakley and D. M. Kienzle. Some Weaknesses of the TCB Model. Panel Debate *In: Proceedings of the 1997 IEEE Symposium on Security and Privacy*, p. 3-5, Oakland, CA, May 1997.

[Blakley 1996] Bob Blakley. The Emperor's Old Armor. *In: Proceedings of the 1996 ACM New Security Paradigm Workshop*, p. 2-16, 1996.

[Boeyen, Howes e Richard 1999] S. Boeyen, T. Howes and P. Richard. Internet X.509 Public Key Infrastructure Operational Protocols - LDAPv2. *Internet Draft RFC 2559*, April 1999. (http://www.ietf.org/rfc/rfc2559.txt)

[Brewer e Nash 1989] D.F.C. Brewer and M.J. Nash. The Chinese Wall Security Policy. *In: Proceedings of the 1989 IEEE Symposium on Security and Privacy*, p. 215-228, Oakland, CA, May 1989.

[Brose 2000] Gerald Brose. A typed access control model for CORBA. *In: Proceedings of the ESORICS 2000 - European Symposium on Research in Computer Security*, p. 88-105, Toulouse, France, 2000, Springer-Verlag LNCS 1895.

[Brose 1998] Gerald Brose. Towards an Access Control Policy Specification Language for CORBA. *In: Proceedings of the ECOOP/EWDOS 1998 - Workshop on Distributed Object Security*, Brussels, Belgium, 1998 (http://sirac.inrialpes.fr/~jensen/ewdos-program.html).

[Brose 1997] Gerald Brose. JacORB – A free Java ORB. *Technical Report B-97-02*, Freie Universität Berlin, Institut für Informatik, Berlin, April 1997 (http://www.inf.fu-berlin.de/~brose/jacorb/).

[Campbell e Qian 1998] Roy H. Campbell and Qian Tin, Dynamic Agent-Based Security Architecture for Mobile Computers, *In: Proc. of the Second PDCN '98*, Australia, December 1998 (http://www.jaist.ac.jp/~hori/conf/PDCN98.html).

[Caplan e Sanders 1999] Kimberly Caplan and James L. Sanders. Building na International Security Standard. *IEEE ITProfessional*, v. 1, n. 2, p. 29-34, 1999.

[Carzaniga, Picco e Vigna 1997] Antonio Carzaniga, Gian Pietro Picco and Giovanni Vigna. Designing Distributed Applications with Mobile Code Paradigms. *In: Proceedings of the 19th International Conference on Software Engineering*, p. 22-32, ACM Press, Boston, 1997. (http://www.cs.ucsb.edu/~vigna/listpub.html)

[CEMEB 1999] Common Evaluation Methodology (CEM) Editorial Board (CEMEB). *Common Methodology for Information Technology Security Evaluation, Part 2: Evaluation Methodology*, CEM-99/045, Version 1.0, August 1999 (http://csrc.nist.gov/cc/cem/cemlist.htm).

[CEMEB 1997] Common Evaluation Methodology (CEM) Editorial Board (CEMEB). *Common Methodology for Information Technology Security Evaluation, Part 1: Introduction and general model*, CEM-97/017, Draft Version 0.6, 1997 (http://csrc.nist.gov/cc/cem/cemlist.htm).

[Chizmadia 2000] D. Chizmadia. An Introduction to the Security Specifications of the Object Management Group, ppt slides. *In: Proceedings of the Fourth DOCsec 2000 - Distributed Object Computing Security Workshop*, April 2000, U.S.A (http://www.omg.org/meetings/docsec/).

[Clark e Wilson 1987] David Clark and David Wilson. A Comparison of Commercial and Military Computer Security Policies. *In: Proceedings of the 1987 IEEE Symposium on Security and Privacy*, p. 184-194, Oakland, CA, May 1987.

[Concept5 1998] CONCEPT FIVE Technologies Inc. New Ways of Thinking: CORBA Security Services in Distributed System Environments. White Paper, 1998. (http://www.concept5.com/docs/wpcorsec.pdf)

[Coulouris 1994] George Coulouris and Jean Dollimore. *Distributed Systems - Concepts and Design*. Addison-Wesley, 1994.

[CSC 1998] CSC - Computer Sciences Corporation. *Cisco Private Internet Exchange (PIX) Firewall 520 Version 4.3(1) Security Target*, Version 1.2, December 1998. (http://niap.nist.gov/cc-scheme/ValidatedProducts.html)

[Dalton e Griffin 1997] C. I. Dalton and J. F. Griffin. Applying Military Grade Security to the Internet. *In: Proceedings of the 8th Joint European Network Conference*, p. 711-1 – 711-8, Edinburgh, 1997.

[Denning 1977] D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, v. 20, n. 7, p. 504-513, July 1977.

[Denning 1976] Dorothy E. Denning. A Lattice Model of Secure Information Flow. *Communications of the ACM*, v. 19, n. 5, p. 236-243, May 1976.

[Deswarte 1991] Y. Deswarte. *Construction des Systèmes d'Exploitation Répartis*, chapter Tolérance aux Fautes, Sécurite et Protection, p. 1-50, Collection Didactique, INRIA, 1991.

[Dierks e Allen 1999] T. Dierks and C. Allen. The TLS Protocol Version 1.0. *RFC2246*, January 1999. (http://www.ietf.org/rfc/rfc2246.txt)

[Dod 1985] Department of Defense Trusted Computer System Evaluation Criteria. DoD 5200.28-STD, December 1985. (http://www.morehouse.org/hin/rainbow/5200_28-STD.txt)

[ECMA235 1996] ECMA. *Standard ECMA-235 – The ECMA GSS-API Mechanism*, ECMA, March 1996. (http://www.ecma.ch/ecma1/STAND/ECMA-235.HTM)

[ElGamal 1985] T. ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, *IEEE Transactions on Information Theory*, v. IT-31, n. 4, p. 469-472, 1985.

[Eriksson e Penker 1998] Hans-Eric Eriksson and Magnus Penker. *UML Toolkit*. John Wiley and Sons, 1998.

[Evans e Rogers 1997] Eric Evans and Daniel Rogers. Using Java applets and CORBA for multi-user distributed applications. *IEEE Internet Computing*, v. 1, n. 3, p. 43-55, May/June 1997.

[Ferraiolo, Barkley e Kuhn 1999] David F. Ferraiolo, John F. Barkley and D. Richard Kuhn. A role based access control model and reference implementation within a corporate intranet. *ACM Transactions on Information and Systems Security*, vol. 2, n. 1, February 1999.

[Ferraiolo, Gilbert e Lynch 1993] David F. Ferraiolo, Dennis M. Gilbert, and Nickilyn Lynch. An Examination of federal and commercial access control policy needs. *In Proceedings of NIST-NCSC National Computer Security Conference*, p. 107-116, Baltimore, MD, September 20-23, 1993.

[Ferraiolo e Kuhn 1992] David F. Ferraiolo and Richard Kuhn. Role-Based Access Control. *In: Proceedings of 15th NIST-NCSC National Computer Security Conference*, p. 554-563, Baltimore, MD, October 13-16, 1992.

[Foley 1998] S. Foley. A Kernelized Architecture for Multilevel Secure Application Policies. *In: Proceedings of the ESORICS - European Symposium on Research in Computer Security*, Louvain-la-Neuve, Belgium, September 1998. (http://www.laas.fr/~esorics/esorics_98/esorics_98.html)

[Foley, Gong e Qian 1996] S. Foley, L. Gong and X. Qian. A security model of dynamic labeling providing a tiered approach to verification. *In Proc. of the 1996 IEEE Symposium on Security and Privacy*, p. 142-153, Oakland, CA, May 1996.

[Foley 1994] S. Foley. Reasoning About Confidentiality Requirements. *In Proceedings of the 7th IEEE Computer Security Foundations Workshop*, p. 150-160, Franconia, New Hampshire, June 1994.

[Ford 1994] Warwick Ford. *Computer Communications Security - Principles, Standard Protocols and Techniques*. Prentice-Hall, 1994.

[Fraga 1985] Joni da Silva Fraga. La Sécurité des Donnés par la Tolérance aux Intrusions. *PhD thesis*, Institut National Polytechnique de Toulouse, 1985.

[Fraim 1983] L. J. Fraim. SCOM: A Solution to the Multilevel Security Problem. IEEE Computer, v. 16, n. 7, p. 26-34, July 1983.

[Freier, Karlton e Kocher 1996] A. O. Freier, P. Karlton and P. C. Kocher. Secure Socket Layer 3.0. Internet Draft, November 1996.

[Garber 2000] Lee Garber. Denial-of-Service Attacks Rip the Internet. *IEEE Computer*, New York, v. 33, n. 4, p. 12-17, April 2000.

[Garfinkel e Spafford 1997] Simson Garfinkel and Gene Spafford. *Web Security & Commerce*. O'Reilly & Associates Inc, 1997.

[Ghezzi e Vigna 1998] Carlo Ghezzi and Giovanni Vigna. Software Engineering Issues in Network Computing, *In M. Broy and B. Rumpe Eds., Requirements Targeting Software and System Engineering, Lecture Notes in Computer Science*, v. 1526, Springer-Verlag, August 1998.

[Giuri 1998] Luigi Giuri. Role-Based Access Control in Java. *In: Proceedings of 3rd ACM Workshop on Role-Based Access*, p. 91-100, Fairfax, VA, October 22-23, 1998. (http://www.acm.org/pubs/contents/proceedings/commsec/286884/index.html)

[Gladney 1997] H. M. Gladney. Access Control for Large Collections. *ACM Transactions on Information Systems*, v. 15, n. 2, p. 154–194, April 1997.

[Goguem e Meseguer 1984] J. Goguem and J. Meseguer. Unwinding and Inference Control. *In: Proceedings of the 1984 IEEE Symposium on Security and Privacy*, p. 75-86, Oakland, California, May 1984.

[Gollmann 2000] Dieter Gollmann. *Computer Security*. John Wiley and Sons, 1999.

[Gong 1989] Li Gong. A Secure Identity-Based Capability Systems. *In Proc. of the 1989 IEEE Symposium on Security and Privacy*, p. 56-63, Oakland, California, May 1989.

[Graz 1999a] Graz - University of Technology. iSaSiLk Toolkit, *Inst. for Applied Information Processing and Communications*, Graz University of Technology, 1999 (http://jcewww.iaik.at/products/isasilk/index.htm).

[Graz 1999b] Graz - University of Technology, *IAIK-JCE 2.5 - Java Cryptography Extensions User Manual*, Institute for Applied Information Processing and Communications, Graz - University of Technology, 1999.

[Hagimont, Huet e Mossière 1997] D. Hagimont, O. Huet, J. Mossière. A protection scheme for a CORBA environment. *In: Proceedings of the ECOOP'97 Workshop-CORBA: Implementation, Use and Evaluation*, Finland, June 1997. (http://ballesta.inrialpes.fr/~bellissa/wecoop97/prev_index.html).

[Harrison 1976] M. A. Harrison, W. L. Ruzzo, J. D. Ullman. Protection in operating systems. *Communications of the ACM*, v. 19, n. 8, p. 461-471, October 1976.

[Hassler 1999] Vesna Hassler. X.500 and LDAP Security: A Comparative Overview. *IEEE Internet Network*, p. 54-64, November/December 1999.

[Hayton, Bacon e Moody 1998] R. J. Hayton, J. M. Bacon and K. Moody. Access Control in an Open Distributed Environment, *In: Proc. of the 1998 IEEE Symposium on Security and Privacy*, p. 3-14, California, May 1998.

[Housley 2000] Russell Housley – Spyrus Chief Scientist. Banking Industry view of Common Criteria. Transparências *In: ICCC First International Common Criteria Conference*, 23-25 May 2000, Baltimore, Maryland, U.S.A. (http://niap.nist.gov/cc-scheme/iccc/trackc.html)

[Iona 1999] Iona Technologies. *Orbix Security 3.0 – White Paper*, September 1999. (http://www.iona.com/support/whitepapers/download.html)

[NSA 1999] NSA National Security Agency. *Information Systems Security Organization - Labeled Security Protection Profile*. Version 1.b, 8 October 1999. (http://www.radium.ncsc.mil/tpep/library/protection_profiles/index.html)

[IBM 1999] IBM Corporation. *IBM Visual Age for Java – Iniciando – Versão 3.0*. 1999.

[ISO/IEC 15408-1 1999] ISO/IEC. 15408-1:1999(E) - Part 1: Introduction and general model. *In: Information Technology – Security Techniques - Evatuation Criteria for IT Security*, First edition, ISO/IEC, December 1999.

[ISO/IEC 15408-2 1999] ISO/IEC. 15408-2:1999(E) - Part 2: Security Functional Requirements. In: *Information Technology – Security Techniques - Evatuation Criteria for IT Security*, First edition, ISO/IEC, December 1999.

[ISO/IEC 15408-3 1999] ISO/IEC. 15408-3:1999(E) - Part 3: Security Assurance Requirements. In: *Information Technology – Security Techniques - Evatuation Criteria for IT Security*, First edition, ISO/IEC, December 1999.

[ITU-TX509 1993] ITU-T. *Information Technology - The Open Systems Interconnection - The Directory: Autentication Framework.* ITU-T Recommendation X.509, November 1993.

[Jajodia e Kogan 1990] S. Jajodia and B. Kogan. Integrating an object-oriented data model with multilevel security. *In: Proceedings of the 1990 IEEE Symposium on Security and Privacy*, p. 76-85, Oakland, California, May 1990.

[Jalote 1991] Pankaj Jalote. *An Integrated Approach to Software Engineering*, Springer-Verlag New York Inc., ISBN 3-540-97561-6, 1991.

[Johner, Brown, Hinner, *et al.* 1998] Heinz Johner, Larry Brown, Franz-Stefan Hinner, Wolfgang Reis, Johan Westman. *Understanding LDAP.* IBM SG24-4986-00, June 1998. (http://www.reedbooks.ibm.com)

[Karjoth 2000] Günter Karjoth. An Operational Semantics of Java 2 Access Control. *In: Proceedings of the IEEE Computer Security Foundations Workshop*, Cambridge, England, July 2000 (http://www.computer.org/proceedings/csfw/0671/0671toc.htm).

[Karjoth 1998] Günter Karjoth. Authorization in CORBA Security. *In: Proceedings of the Fifth ESORICS*, Lecture Notes in Computer Science, p. 143-158, Springer-Verlag, Berlin Germany, September 1998.

[Kent 1999] Stephen Kent. Network Security: Then and Now or 20 Years in 10 Minutes. *In Proc. of the 1999 IEEE Symposium on Security and Privacy, Panel 1: Brief History of Twenty Years of Computer Security Research,* Oakland, California, May 1999.

[Kent 1977] Stephen Kent. Encryption-Based Protection for Interactive User/Computer Communication. *In Proceedings of the Fifth Data Communications Symposium*, IEEE Computer Press, New York, September 1977.

[Klevinsky 1999] T. J. Klevinsky, Contemporary Hacking Tools and Their Use in Penetration Testing. *In: FCSC99 - The Federal Computer Security Conference*. Course Day, May 1999, Baltimore, MD, U.S.A (http://www.cio.org/sans99/).

[Kohl e Neuman 1993] J. Kohl and C. Neuman. The Kerberos Network Authentication Service (V5), *Internet Draft RFC1510*, September 1993 (http://www.ietf.org/rfc/rfc1510.txt).

[Lai 1992] X. Lai. *On the Design and Security of Block Ciphers*, ETH Series in Information Processing, v. 1, Konstanz:Hartung-Gorre Verlag, 1992.

[Lampson 1971] Butler W. Lampson. Protection. In: *5th Princeton Symposium on* Information *Sciences and Systems*, March 1971. Reprinted in *ACM Operating Systems Review*, v. 8, n.1, p. 18-24, 1974.

[Lampson 1973] Butler W. Lampson. A Note on the Confinement Problem. *Communications of the ACM*, v. 16, n. 10, p. 613-615, October 1973.

[Landwehr, Heitmeyer e McLean 1984] C. E. Landwehr, C. L. Heitmeyer and J. McLean. A Security Model for Military Message Systems. *ACM Transactions on Computer Systems*, v. 9, n. 3, p. 198-222, August 1984.

[Landwehr 1983] C. E. Landwehr. The best available technologies for computer security. *ACM Computing Surveys ACM*, v. 16, n. 7, p. 86-95, July 1983.

[Landwehr 1981] C. E. Landwehr. Formal models for computer security. *Computing Surveys ACM*, v. 13, n. 3, p. 247-278, September 1981.

[Lang 1999] Ulrich Lang. *Distributed Access Control*, PhD Thesis Proposal, 1999. (http://www.cl.cam.ac.uk/~ul201/proposal.pdf)

[Lang 1997] Ulrich Lang. *Security Aspects of the Common Object Request Broker Architecture*, University of London, 1997, MSc Information Security Thesis. (http://www.cl.cam.ac.uk/~ul201/mscdissertation.pdf)

[Lang e Schreiner 2000] Ulrich Lang and Rudolf Schreiner. Flexibility and Interoperability in CORBA Security. Eletronic Notes in Theoretical Computer Science, v. 32, Elsevier Science B. V., 2000.
(http://www.elsevier.nl/gej-ng/31/29/23/57/show/Products/notes/cover.htt)

[LaPadula e Bell 1996] Leonard J. LaPadula and D. Elliot Bell. Mitre Technical Report 2547 - Volume II. *Journal of Computer Security*, v. 4, issue 2/3, p. 299-323, 1996.

[Lindqvist e Jonsson 1998] Ulf Lindqvist and Erland Jonsson. A Map of Security Risks Associated with Using COTS. *IEEE Computer*, v. 31, n. 6, June 1998.

[Linn 1997] J. Linn. Generic Security Service Application Program Interface, Version 2, *Internet Draft RFC2078*, January 1997. (http://www.ietf.org/rfc/rfc2078.txt)

[Lista corba-security 2000] Lista de discussão sobre o *corba-security* [online]. Disponível: Email: corba-security@cs.fiu.edu [mensagem capturada em 8 fev. 2000]

[Lista corba-security 1999] Lista de discussão sobre o *corba-security* [online]. Disponível: http://www.cs.fiu.edu/~beznosov/corba/security/mail-list/archive/1999/msg00102.html Email: corba-security@cs.fiu.edu [mensagem capturada em 19 aug. 1999].

[Markham, Colby e Denz 1999] Tom Markham, Dwight Colby and Mary Denz. Security Modeling in the COTS Environment. *In: Proceedings of the ACM New Security Paradigms Workshop*, p. 96-102, Caledon Hills, Canada, 1999.

[Mayfield, Roskos, Welke *et al.* 1991] Terry Mayfield, J. Eric Roskos, Stephen R. Welke and John M. Boone. Integrity in Automated Information Systems. *NCSC Technical Report 79-91*, Institute for Defense Analysis, Alexandria, VA, September 1991.

[McGraw e Felten 1997] Gary McGraw and Edward W.Felten. *Java Security : Hostile Applets, Holes and Antidotes.* John Wiley and Sons, 1997.

[McLean 1994] John McLean. *Security Models*. In: Encyclopedia of Software Engineering, Wiley & Sons, 1994 (http://chacs.nrl.navy.mil/personnel/mclean-txt.html).

[McLean 1990] John McLean. The Specification and Modeling of Computer Security. *IEEE Computer*, v. 23, n. 1, p. 9-16, January 1990.

[McLean 1987] John McLean. Reasoning About Security Models. *In: Proceedings of the 1987 IEEE Symposium on Security and Privacy*, p. 123-131, Oakland, California, April 1987.

[Millen 1999] Jonathan Millen. 20 Years of Covert Channel Analysis. *In: Proceedings of the 1999 IEEE Symposium on Security and Privacy, Panel 1: Brief History of Twenty Years of Computer Security Research*, Oakland, California, May 1999.

[Millen e Lunt 1992] J. K. Millen and T. F. Lunt. Security for object-oriented database systems. *In: Proceedings of the 1992 IEEE Symposium on Security and Privacy*, p. 260-272, Oakland, California, May 1992.

[Miller, Neuman, Schiller *et al.* 1988] S.P. Miller, B. C. Neuman, J. I. Schiller and J.H. Saltzer. Section E.2.1: Kerberos Authentication and Authorization System. *Project Athena Technical Plan*, MIT Project Athena, Cambridge, Massachusetts, October 1988.

[Módulo 1999] Módulo Security Solutions S.A. *5ª Pesquisa Nacional sobre Segurança da Informação*. 1999. (http://www.modulo.com.br/)

[Morrissey 1998] John Morrissey. *Security Target for Oracle7 Database Server Release 7.3.4*, June 1998. (http://niap.nist.gov/cc-scheme/ValidatedProducts.html)

[Nagaratnam 1998] Nataraj Nagaratnam. Role-Based Protection and Delegation for Mobile Object Environments. *In: Proceedings of the 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations in Association with the 12th European Conference on Object-Oriented Programming (ECOOP'98)*, Brussels, Belgium, 21 July 1998. (http://cuiwww.unige.ch/~ecoopws)

[Nephilim 1999] *NEPHILIM: Java Implementation of CORBA Security Services*. University of Illinois, 1999. (http://choices.cs.uiuc.edu/Security/nephilim)

[Neuman 1994] B. Clifford Neuman. *Scale in Distributed Systems*. Chapter In: Readings in Distributed Systems. IEEE Computer Society Press, ISBN 0818630329, January 1994.

[Neuman e Ts'o 1994] B. Clifford Neuman and Theodore Ts'o. Kerberos: An Authentication Service for Computer Networks, *IEEE Communications*, v. 32, n. 9, p. 33-38, September 1994.

[Nicomette 1997] Vincent Nicomette and Yves Deswarte. An Authorization Scheme for Distributed Object Systems. *In: Proc. of the IEEE Symposium on Research in Security and Privacy*, p. 21-30, Oakland, California, 1997.

[Nicomette 1996] Vincent Nicomette. *La Protection dans les Systèmes à Objets Répartis*. PhD thesis, Institut National Polytechnique de Toulouse, 1996.

[Nikander e Partanen 1999] Pekka Nikander and Jonna Partanen. Distributed Policy Management for JDK 1.2, *In: Proceedings of the 1999 Network and Distributed Systems Security Symposium*, 3-5 February 1999, San Diego, California, Internet Society, February 1999. (http://www.isoc.org/isoc/conferences/ndss/99/proceedings/)

[NIST/NSA 1999] NIST and NSA. *Common Criteria Evaluation and Validation Scheme for Information Technology Security Organization, Management and Concepts of Operations*, Scheme Publication #1, Version 2.0, May 1999. (http://niap.nist.gov/cc-scheme/GuidanceDocs.html)

[NIAP/NIST 1998] Common Criteria Project. *Arrangement on the Mutual Recognition of Common Criteria Certificates in the field of IT Security*. 5 October 1998. (http://niap.nist.gov/cc-scheme/SchemeDoc/ccmra-v1.pdf)

[Oaks 1999] Scott Oaks. *Segurança de Dados em Java*. Editora Ciência Moderna Ltda, Rio de Janeiro, 1999.

[Obelheiro 2000] Rafael Rodrigues Obelheiro. Controle de Acesso Baseado em Papéis, *Relatório Técnico 01-2000*, UFSC-DAS-CPGEEL.

[ObjectSpace 1999] Object Space, *VoyagerSecurity 3.2 – Developer Guide*, 1999. (http://www.objectspace.com/products/documentation/voyager_html_docs/security/index.htm)

[OMG 2000a] Object Management Group. Security Service:v1.5, *OMG Document Number 00-06-25*, June 2000 (ftp://ftp.omg.org/pub/docs/formal/00-06-25.pdf).

[OMG 2000b] Object Management Group. Common Secure Interoperability Version 2 – Final Submission, *OMG Document Number orbos/00-08-04*, August 2000.

[OMG 2000c] Object Management Group. ATLAS: The Authorization Token Layer Acquisition Service - Common Security Interoperability Version 2 – Joint Partial Submission, *OMG Document Number orbos/00-08-05*, August 2000.

[OMG 2000d] DSTC. Public Key Infrastructure RFP - Revised Submission. *OMG Document Number ec/*99-12-03, December 1999 (ftp://ftp.omg.org/pub/docs/ec/99-12-03.pdf).

[OMG 2000e] Object Management Group. Joint Revised Submission CORBA/Firewall Security, *OMG Document Number orbos/98-05-04*, May 1998. (ftp://ftp.omg.org/pub/docs/orbos/98-05-04.pdf)

[OMG 2000f] Object Management Group. Security Domain Membership Management Service – Revised Submission, *OMG Document Number orbos/00-02-04*, 14 February 2000. (http://www.omg.org/techprocess/meetings/schedule/Security_Domain_Membership_RFP.html)

[OMG 1999a] Object Management Group. ORB Interface, *OMG Document Number 99-07-08*, June 1999.

[OMG 1999b] Object Management Group. CORBA 2.3.2 Specification, *OMG Document Number 99-10-07*, October 1999. (http://www.omg.org/technology/documents/formal/corba2formal.htm)

[OMG 1999c] Object Management Group. Common Secure Interoperability Version 2, *OMG Document Number orbos/99-01-10*, January 1999. (http://www.omg.org/techprocess/meetings/schedule/Common_Secure_Interop._V2_RFP.html)

[Opengroup 2000] Opengroup. *DCE Portal.* (http://www.opennc.org/dce/)

[Orfali e Harkey 1997] Robert Orfali and Dan Harkey. *Client/Server Programming with JAVA and CORBA*. John Wiley & Sons, 1997.

[Orfali, Harkey e Edwards 1997] Robert Orfali, Dan Harkey, and Jeri Edwards. The Web is in trouble. CORBA and Java are out to save it. *Byte Magazine*, October 1997. (http://byte.com/art/9710/sec6/art3.htm#107fe1c2)

[Osborn, Sandhu e Munawer 2000] Sylvia Osborn, Ravi S. Sandhu and Qamar Munawer. Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies. *ACM Transactions on Information and Security Systems*, v.3, n.2, May 2000.

[Osborn 1997] Sylvia Osborn. Mandatory Access Control and Role-Based Access Control Revisited. *In: Proceedings of the 2nd ACM Workshop on Role-Based Access Control*, p. 31-40, Fairfax, VA, 1997.

[Pfleeger 1996] Charles P. Pfleeger. *Security in Computing.* Prentice-Hall, 1996.

[Pistola, Reller, Gupta, *et al.* 1999] Marco Pistola, Duane F. Reller, Deepak Gupta, Milind Nagnur and Ashok K. Ramani. *Java 2 Network Security.* IBM Redbooks SG24-2109-01, IBM Corporation - International Technical Support Organization, Second edition, June 1999 (http://www.redbooks.ibm.com).

[Promia 1999] Promia Inc. *SecureBroker* (http://www.promia.com/products/securebroker.html).

[Ramaswamy e Sandhu 1998] Chandramouli Ramaswamy and Ravi S. Sandhu. Role-Based Access Control Features in Commercial Database Management Systems. *In Proc. of the 21st NISSC National Information Systems Security Conference*, Crystal City, Virginia, USA, October, 1998. (http://csrc.nist.gov/nissc/1998/papers.html)

[Resnick 1996] Ron I. Resnick. Distributed Objects on the WWW: A Position Paper. *In: Proceedings of the OOPSLA96 (The Eleventh Annual ACM Conference on Object-Oriented Programming) Workshop - Toward the Integration of WWW and Distributed Object Technology*, October, 1996 (http://www.eng.uci.edu/~peilei/index.html)

[Rivest 1995] R. L. Rivest. The RC5 Encryption Algorithm, *Dr. Dobb's Journal*, v. 20, n. 1, p. 146-148, January 1995.

[Rivest, Shamir e Adleman 1978] R. L. Rivest, A. Shamir, L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, *Communications of the ACM*, v. 21, n. 2, p. 120-126, February 1978.

[Rubin, Geer e Ranum 1997] Aviel D. Rubin, Dan Geer and Marcus Ranum. *Web Security Sourcebook*. John Wiley & Sons, 1997.

[Ruh, Herron e Klinker 1999] William Ruh, Thomas Herron and Paul Klinker. *IIOP Complete – Understanding Middleware Interoperability*, The Addison-Wesley Object Technology Series, 1999.

[Ryan e Seligman 1999] V. Ryan and S. Seligman. Schema for Representing CORBA Object References in an LDAP Directory. *RFC2714*, October 1999. (http://www.ietf.org/rfc/rfc2714.txt)

[Santin 2000] Altair Olivo Santin. Estudo de Casos: LDAP, *Relatório Técnico 02-2000*, UFSC-DAS-CPGEEL.

[Sandhu, Ferraiolo e Kuhn 2000] Ravi S. Sandhu, David Ferraiolo and Richard Kuhn. The NIST Model for Role-Based Access Control: Towards a Unified Standard. *In Proc. of the 5th ACM/RBAC Conference*, 2000. (http://www.acm.org/sigsac/rbac2000.html)

[Sandhu 1998] Ravi S. Sandhu. Role-Based Access Control. In Selkowitz editor, *Advances in Computers*, v. 46, Academic Press, 1998.

[Sandhu, Coyne, Feinstein, *et al.* 1996] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, v. 29, n. 2, p. 38-47, February 1996.

[Sandhu e Samarati 1994] Ravi S. Sandhu and Pierangela Samarati. Access control: Principles and practice. *IEEE Communications*, v. 32, n. 9, p. 40-48, 1994.

[Sandhu 1993] Ravi S. Sandhu. Lattice-based access control models. *IEEE Computer*, v. 26, n. 11, p. 9-19, November 1993.

[Sans 2000] Sans Institute Resources. *How to Eliminate the Ten Most Critical Internet Security Threats - The Experts' Consensus.* Version 1.22, June 19, 2000. (http://www.sans.org/topten.htm)

[Schneier 1996] Bruce Schneier. *Applied Cryptography.* 2nd edition, John Wiley & Sons, 1996.

[Siegel 1998] Jon Siegel. OMG Overview: CORBA and OMA in Enterprise Computing. *Communications of the ACM*, v. 41, n. 10, October 1998.

[SilverStream 2000] SilverStream Inc. *Jbroker 3.0 Tutorial.*
(http://www1.silverstream.com/jbrokerdoc/docs/tutorial/index.html)

[Smid e Branstad 1988] M. E. Smid and D. K. Branstad. The Data Encryption Standard: Past and Future, *In: Proceedings of the IEEE*, v. 76, n. 5, p. 550-559, May 1988.

[Snyder 1981] L. Snyder. Theft and conspiracy in the take-grant protection model. *Journal of Computer and System Sciences*, v. 23, n. 3, p. 333-347, December 1981.

[Srinivas 2000] Raghavan N. Srinivas. *Java security evolution and concepts, Part 2 - Discover the ins and outs of Java security.* JavaWorld Online Magazine, August 2000.
(http://www.javaworld.com/javaworld/jw-07-2000/jw-0728-security.html)

[Stallings 1998] William Stallings. *Cryptography and Network Security : Principles and Practice.* 2nd edition, Prentice-Hall, July 1998.

[Steiner, Neuman e Schiller 1988] J. G. Steiner, B. C. Neuman and J. I. Schiller. Kerberos: An authentication service for open network systems. *In: Proceedings of the 1988 Usenix Conference*, p. 191-201, February 1988.

[Sun 1995a] Sun Microsystems Inc. *The Java Language Specification.* Sun Microsystems Inc., Mountain View, CA, October 1995.

[Sun 1995b] Sun Microsystems Inc., *The Java Language Environment : A White Paper.* Sun Microsystems Inc., Mountain View, CA, 1995.

[Sun 1996] Sun Microsystems Inc. *The Java Platform : A White Paper.* Sun Microsystems Inc., Mountain View, CA, May 1996.
(http://java.sun.com/docs/white/platform/CreditsPage.doc.html)

[Sun 1997a] Sun Microsystems Inc. Secure Computing with Java : Now and the Future. *In Proceedings of the JavaOne Conference*, WhitePaper, 1997. (http://java.sun.com/security/javaone97-whitepaper.html)

[Sun 1997b] Sun Microsystems Inc. *Applet Security Frequently Asked Questions*. Sun Microsystems Inc., Mountain View, CA, 1997. (http://java.sun.com/sfaq)

[Sun 1997c] Sun Microsystems Inc. *Java Security*. Sun Microsystems White Paper, Mountain View, CA, 1996. (http://java.sun.com/security/index.html)

[Sun 1998] Sun Microsystems Inc. *Java Security Architecture (JDK 1.2)*. Sun Microsystems Inc., Mountain View, CA, October 1998. (http://java.sun.com:80/products/jdk/1.2/docs/guide/security/spec/security-spec.doc.html)

[Sun 1999] Sun Microsystems Inc. *Java Cryptography Architecture API Specification & Reference*. Sun Microsystems Inc., Mountain View, CA, December 1999. (http://java.sun.com:80/products/jdk/1.2/docs/guide/security/CryptoSpec.html)

[Sun 2000] Sun Microsystems Inc. *The Java Tutorial – A practical guide for programmers*. Sun Microsystems Inc., Mountain View, CA, May 2000. (http://java.sun.com/docs/books/tutorial/)

[Thorn 1997] Tommy Thorn. Programming Languages for Mobile Code. *ACM Computing Surveys*, v. 29, n. 3, p. 213-239, September 1997.

[Tipton 1998] Hal Tipton and Micki Krause. *Handbook of Information Security Management*. Consulting Editors, Auerbach Publications, CRC Press LLC, 1998, ISBN: 0849399475. (http://www.itknowledge.com/reference/standard/0849399475/ewtoc.html)

[Troy 1999] Gene Troy. *Introduction to the Common Criteria for IT Security (ISO 15408)*, NIST slides, March 1999 (http://csrc.nist.gov/cc/info/japan-brief-990318.ppt).

[Wahl, Howes e Kille 1997] M. Wahl, T. Howes and S. Kille. Lightweight Directory Access Protocol (v3). *RFC2251*, December 1997. (http://www.ietf.org/rfc/rfc2251.txt)

[Wangham 2000] Michelle Silva Wangham. *Estudo e Implementação de um Esquema de Autorização Discricionária baseado na Especificação CORBAsec*. Florianópolis, 2000. Dissertação (Mestrado em Engenharia Elétrica) – Centro Tecnológico, UFSC.

[Wangham, Lung, Westphall, *et al.* 2000] Michelle Silva Wangham, Lau Cheuk Lung, Carla Merkle Westphall, Joni da Silva Fraga, Ricardo Sangoi Padilha e Luciana Moreira Sá de Souza. Projeto, Implementação e Avaliação do *JaCoWeb Security*. *In: Anais do CLEI (Conferencia Latinoamericana de Informática)*, México, Setembro 2000.

(http://clei2000.org.mx/Jueves.html).

[Westphall e Fraga 1999a] Carla Merkle Westphall and Joni da Silva Fraga. Authorization Schemes for Large-Scale Systems based on Java, CORBA and Web Security Models *In: Proceedings of the ICON 99 - The IEEE International Conference on Networks*, September 28 to October 1 1999, Brisbane-Queensland, Australia, p. 327-334. (http://www.lcmi.ufsc.br/~merkle/publica.html)

[Westphall e Fraga 1999b] Carla Merkle Westphall and Joni da Silva Fraga. A Large-scale System Authorization Scheme Proposal Integrating Java, CORBA and Web Security Models and a Discretionary Prototype. *In: Proceedings of the IEEE LANOMS'99 Latin American Network Operations and Management Symposium*, December 03-05 1999, Rio de Janeiro, Brazil, p. 14-25.

[Westphall, Fraga e Lung 1999c] Carla Merkle Westphall, Joni da Silva Fraga e Lau Cheuk Lung. JaCoWe - Um Esquema de Autorização para Redes de Larga Escala integrando os Modelos de Segurança Java, CORBA e Web. *In: Anais do XXV CLEI (Conferencia Latinoamericana de Informatica)*, 30/08/99 à 03/09/99, Assunção, Paraguai, p. 1025-1036.

[Westphall, Fraga e Wangham 2000] Carla Merkle Westphall, Joni da Silva Fraga e Michelle Silva Wangham. PoliCap - Um Serviço de Política para o Modelo CORBA de Segurança. *In: Anais do 18o Simpósio Brasileiro de Redes de Computadores - SBRC 2000*, 23 a 26 de maio 2000, Minas Gerais, Brasil, p. 355-370.

[Williams 2000] Jim William e Allen Basey. Towards a Common Vulnerability Database For CC Evaluations. Transparências *In: ICCC First International Common Criteria Conference*, 23-25 May 2000, Baltimore, Maryland, U.S.A. (http://niap.nist.gov/cc-scheme/iccc/trackb.html)

[Woodward 1987] J. P. L. Woodward. Exploiting the Dual Nature of Sensitivity Labels. *In: Proc. of the IEEE Symposium on Research in Security and Privacy*, p. 23-30, Oakland, California, 1987.

[Zhong e Edwards 1998] Q. Zhong and N. Edwards. Security Control for COTS Components. *IEEE Computer*, v. 31, n. 6, p. 67-73, June 1998.

[Zurko, Simon e Sanfilippo 1999] Mary Ellen Zurko, Rich Simon and Tom Sanfilippo. A User-Centered, Modular Authorization Service Built on an RBAC Foundation. *In: Proc. of the 1999 IEEE Symposium on Research in Security and Privacy*, Oakland, California, 1999.